Hive实现原理 1

新人培训课程 | 从入门到精通

作者：周忱 | 淘宝综合产品
微博：@MinZhou
邮箱：zhouchen.zm@taobao.com

关于我

- 花名:周忱(chén)
- 真名:周敏
- 微博: @MinZhou
- Twitter: @minzhou
- 2010年6月加入淘宝
- 曾经淘宝Hadoop&Hive研发组Leader
- 目前专注分布式实时计算
- Hive Contributor
- 自由、开源软件热爱者

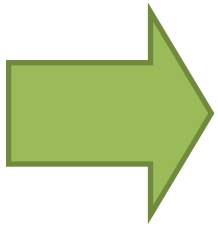
关于我

- 花名:周忱(chén)
- 真名:周敏
- 微博: @MinZhou
- Twitter: @minzhou
- 2010年6月加入淘宝
- 曾经淘宝Hadoop&Hive研发组Leader
- 目前专注分布式实时计算
- Hive Contributor
- 自由、开源软件热爱者

如何用MR实现下面语句?

pv_users

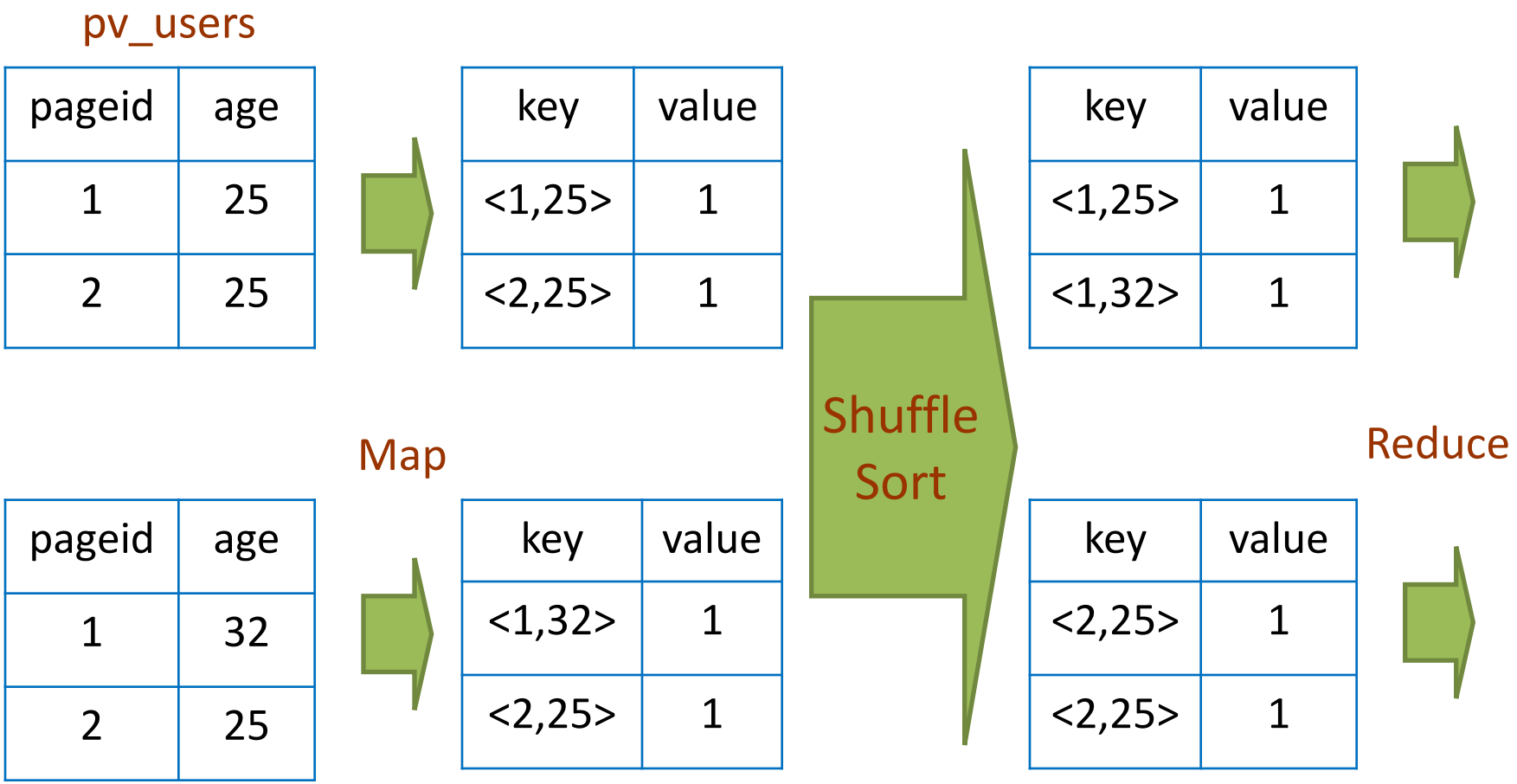
pageid	age
1	25
2	25
1	32
2	25



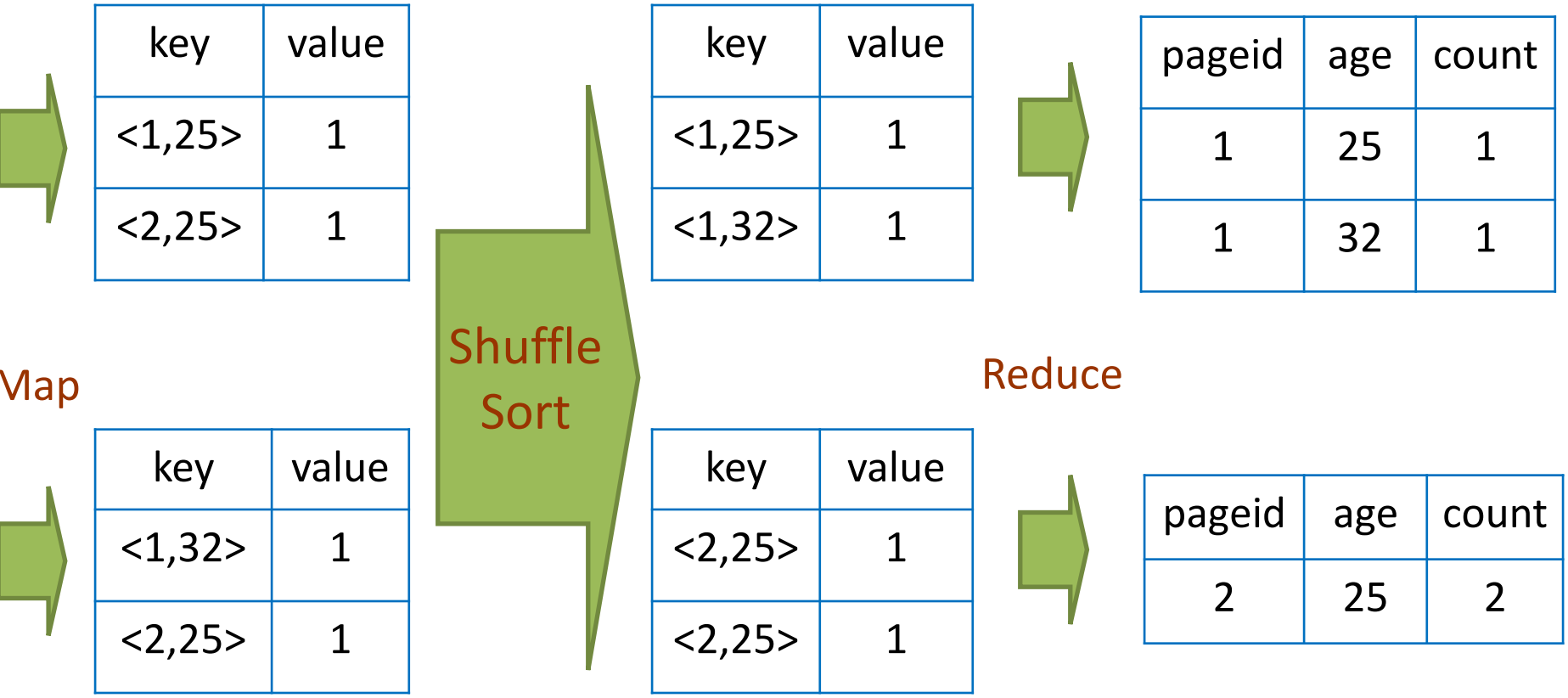
pageid	age	count
1	25	1
2	25	2
1	32	1

```
SELECT pageid, age, count(1)
FROM pv_users
GROUP BY pageid, age;
```

就是这么简单


pv_users
pageid
age
1
25
2
25
Map
pageid
age
1
32
2
25
key
value
<1,25>
1
<2,25>
1
<1,32>
1
<2,25>
1
Shuffle
Sort
key
value
<1,25>
1
<1,32>
1
<2,25>
1
<2,25>
1
Reduce

就是这么简单

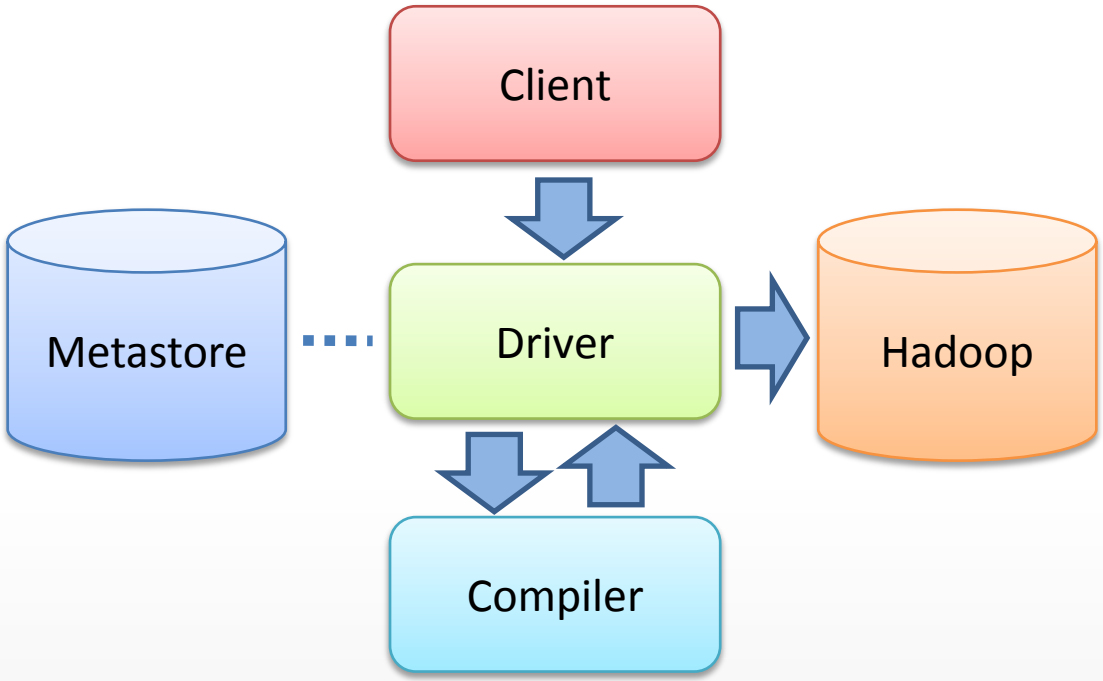

key
value
<1,25>
1
<2,25>
1
key
value
<1,32>
1
<2,25>
1
Map
Shuffle
Sort
key
value
<1,25>
1
<1,32>
1
key
value
<2,25>
1
<2,25>
1
Reduce
pageid
age
count
1
25
1
1
32
1
pageid
age
count
2
25
2

为什么要学习Hive的实现?

- Hive学习曲线平缓,适合非专业人员,集团内部普遍使用
- 一道Hive SQL将转换为多少道M/R作业?
- 我们怎么加快Hive SQL的执行速度?
- 编写Hive SQL的时候我们可以做些什么?
- Hive怎么将HiveQL转换成M/R作业?
- Hive将会采用什么样的优化方式?

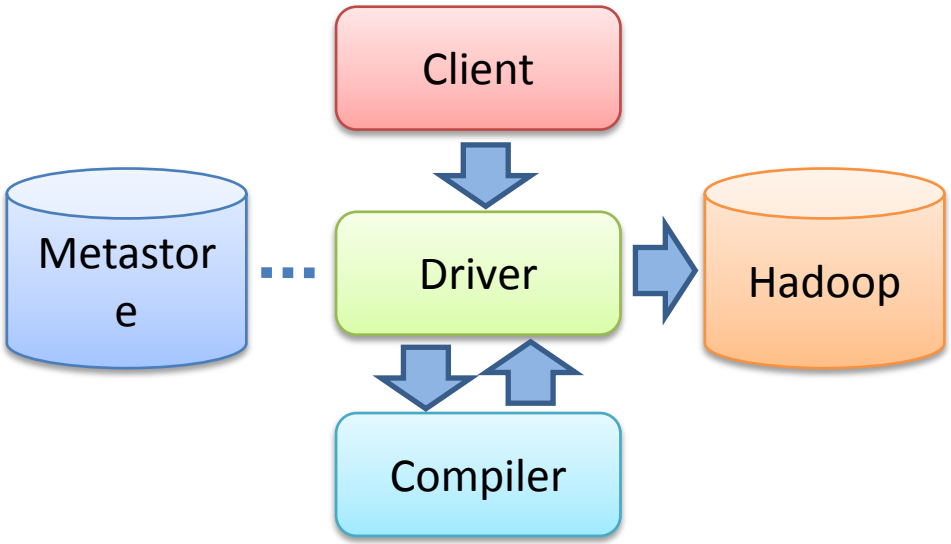
组件分析

Hive架构&执行流程



Hive执行流程

- 编译器将Hive SQL 转换成一组操作符(Operator)
- 操作符是Hive的最小处理单元
- 每个操作符处理代表一道HDFS操作或MapReduce作业



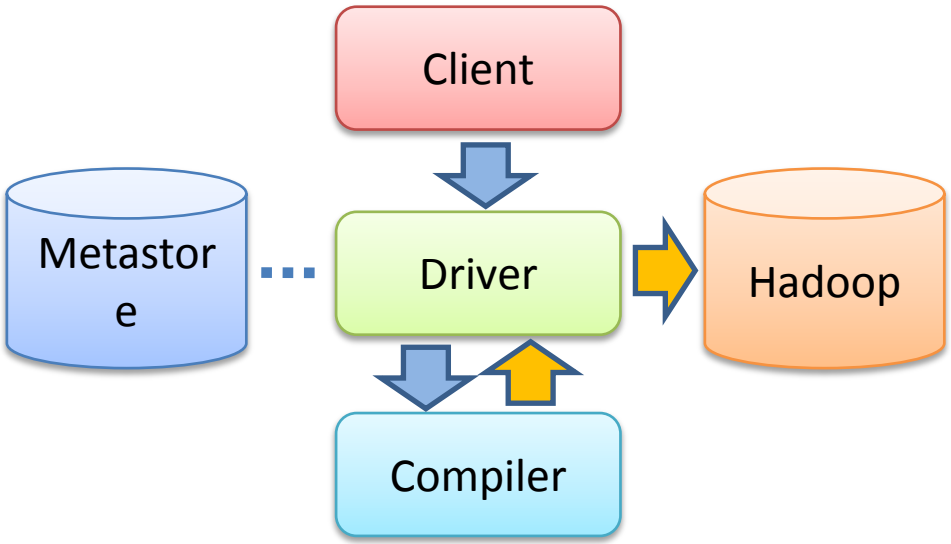
Hive执行流程

- 操作符

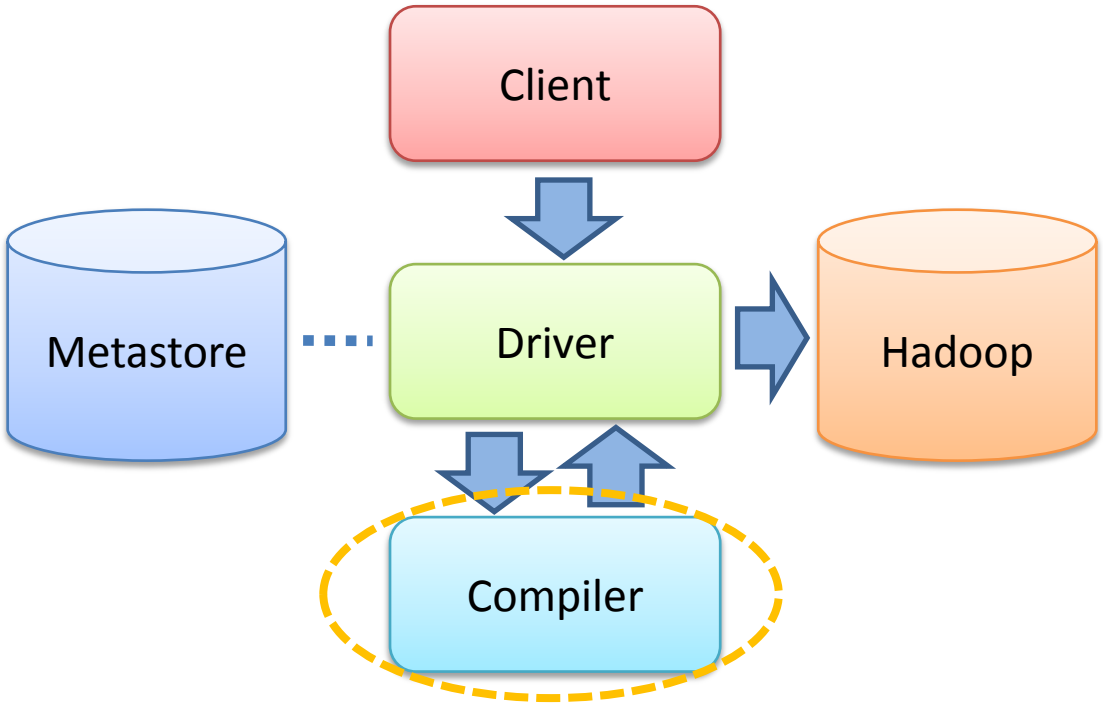
操作符	描述
TableScanOperator	扫描hive表数据
ReduceSinkOperator	创建将发送到Reducer端的<Key,Value>对
JoinOperator	Join两份数据
SelectOperator	选择输出列
FileSinkOperator	建立结果数据,输出至文件
FilterOperator	过滤输入数据
GroupByOperator	Group By语句
MapJoinOperator	/*+ mapjoin(t) */
LimitOperator	Limit语句
UnionOperator	Union语句

Hive执行流程

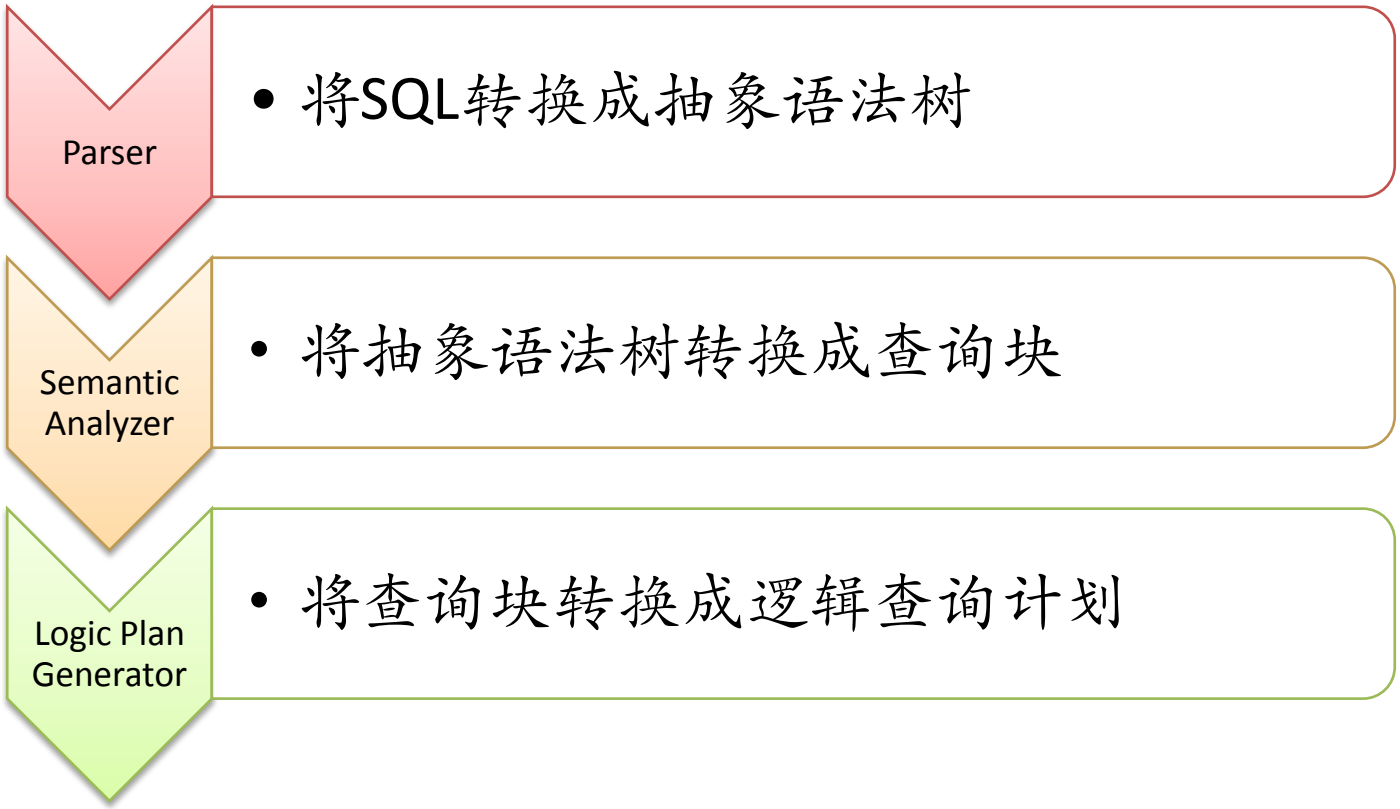
- Hive通过ExecMapper和ExecReducer执行MapReduce任务
- 在执行MapReduce时有两种模式
  - 本地模式
  - 分布式模式



Hive架构&执行流程



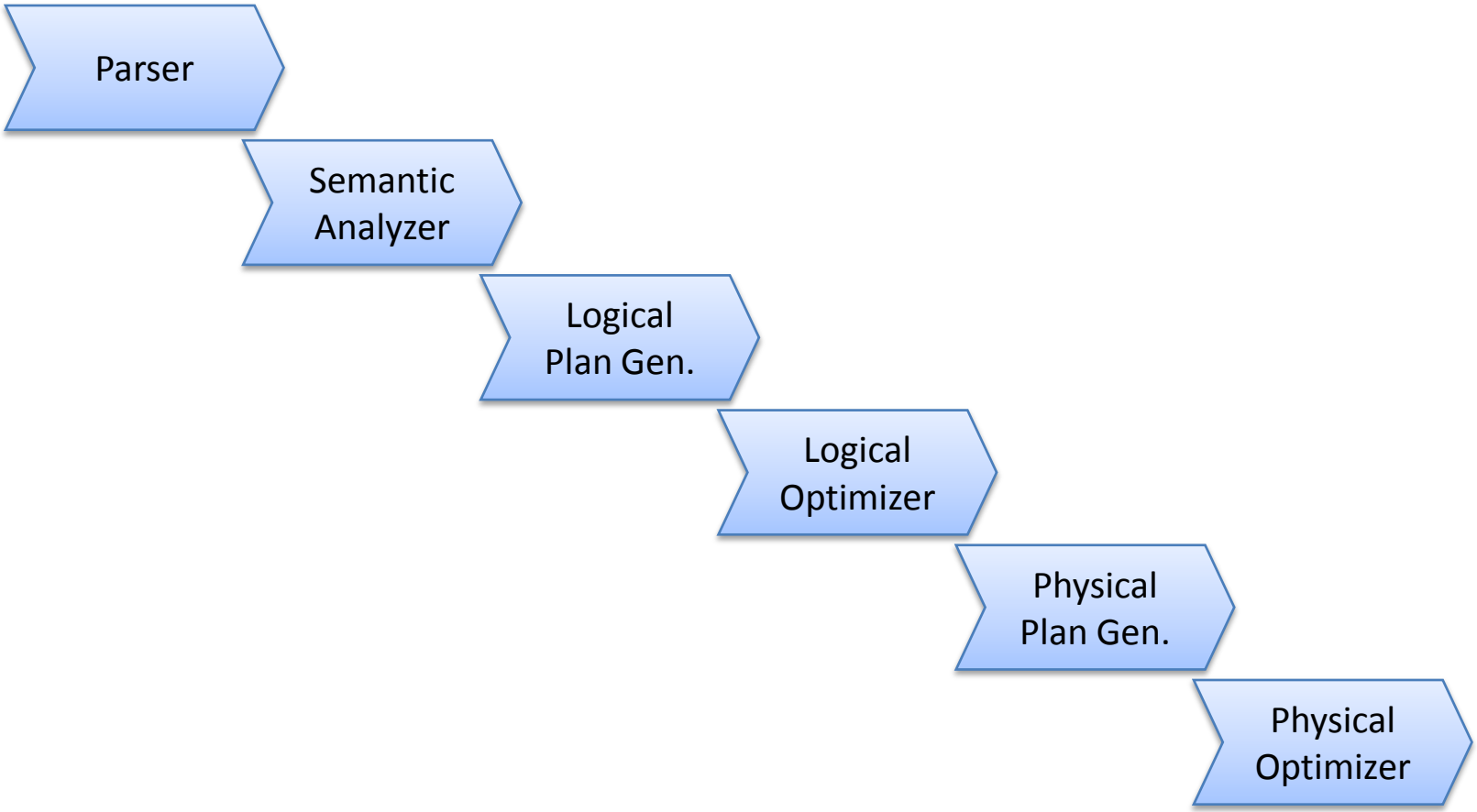
Hive编译器


Parser
• 将SQL转换成抽象语法树
Semantic Analyzer
• 将抽象语法树转换成查询块
Logic Plan Generator
• 将查询块转换成逻辑查询计划

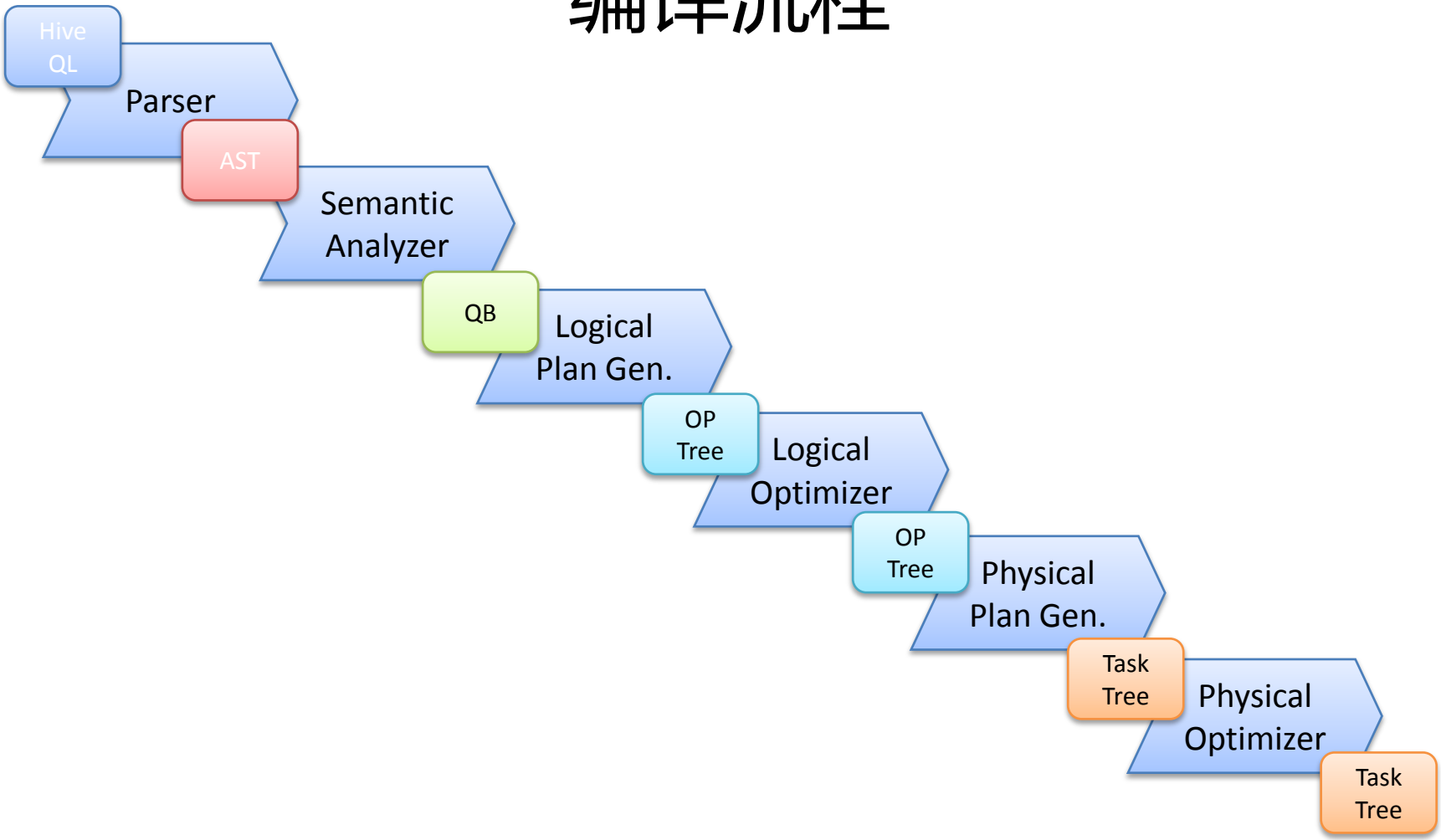
Hive编译器


Logical Optimizer
• 重写逻辑查询计划
Physical Plan Generator
• 将逻辑计划转成物理计划 (M/R jobs)
Physical Optimizer
• 选择最佳的Join策略

编译流程


Parser
Semantic Analyzer
Logical Plan Gen.
Logical Optimizer
Physical Plan Gen.
Physical Optimizer

编译流程


Hive QL
Parser
AST
Semantic Analyzer
QB
Logical Plan Gen.
OP Tree
Logical Optimizer
OP Tree
Physical Plan Gen.
Task Tree
Physical Optimizer
Task Tree

Parser

SQL → AST

Hive QL

```
INSERT OVERWRITE TABLE access_log_temp2
 SELECT a.user, a.prono, p.maker, p.price
 FROM access_log_hbase a JOIN product_hbase p ON (a.prono = p.prono);
```

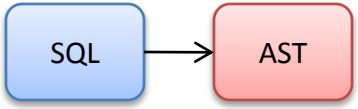
AST

```
TOK_QUERY
 + TOK_FROM
    + TOK_JOIN
       + TOK_TABREF
          + TOK_TABNAME
             + "access_log_hbase"
          + a
       + TOK_TABREF
          + TOK_TABNAME
             + "product_hbase"
          + "p"
       + "="
          + "."
             + TOK_TABLE_OR_COL
                + "a"
             + "access_log_hbase"
          + "."
             + TOK_TABLE_OR_COL
                + "p"
             + "prono“
```

```
+ TOK_INSERT
   + TOK_DESTINATION
      + TOK_TAB
         + TOK_TABNAME
            + "access_log_temp2"
   + TOK_SELECT
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "a"
            + "user"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "a"
            + "prono"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "p"
            + "maker"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "p"
            + "price"
```

Parser → Semantic Analyzer → P → Physical Optimizer

编译流程



SQL

```
INSERT OVERWRITE TABLE access_log_temp2
 SELECT a.user, a.prono, p.maker, p.price
 FROM access_log_hbase a JOIN product_hbase p ON (a.prono = p.prono);
```

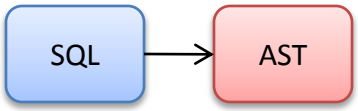
AST

```
TOK_QUERY
 + TOK_FROM
    + TOK_JOIN
       + TOK_TABREF
          + TOK_TABNAME
             + "access_log_hbase"
          + a
       + TOK_TABREF
          + TOK_TABNAME
             + "product_hbase"
          + "p"
       + "="
          + "."
             + TOK_TABLE_OR_COL
                + "a"
             + “prono"
          + "."
             + TOK_TABLE_OR_COL
                + "p"
             + "prono“
```

1

```
+ TOK_INSERT
   + TOK_DESTINATION
      + TOK_TAB
         + TOK_TABNAME
            + "access_log_temp2"
   + TOK_SELECT
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "a"
            + "user"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "a"
            + "prono"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "p"
            + "maker"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "p"
            + "price"
```

编译流程



SQL

```
INSERT OVERWRITE TABLE access_log_temp2
 SELECT a.user, a.prono, p.maker, p.price
 FROM access_log_hbase a JOIN product_hbase p ON (a.prono = p.prono);
```

AST

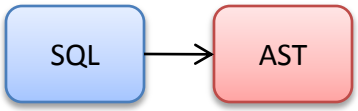
```
TOK_QUERY
 + TOK_FROM
    + TOK_JOIN
       + TOK_TABREF
          + TOK_TABNAME
             + "access_log_hbase"
          + a
       + TOK_TABREF
          + TOK_TABNAME
             + "product_hbase"
          + "p"
       + "="
          + "."
             + TOK_TABLE_OR_COL
                + "a"
             + “prono"
          + "."
             + TOK_TABLE_OR_COL
                + "p"
             + "prono“
```

1

```
+ TOK_INSERT
   + TOK_DESTINATION
      + TOK_TAB
         + TOK_TABNAME
            + "access_log_temp2"
   + TOK_SELECT
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "a"
            + "user"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "a"
            + "prono"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "p"
            + "maker"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "p"
            + "price"
```

2

编译流程



SQL

```
INSERT OVERWRITE TABLE access_log_temp2
 SELECT a.user, a.prono, p.maker, p.price
 FROM access_log_hbase a JOIN product_hbase p ON (a.prono = p.prono);
```

AST

```
TOK_QUERY
 + TOK_FROM
    + TOK_JOIN
       + TOK_TABREF
          + TOK_TABNAME
             + "access_log_hbase"
          + a
       + TOK_TABREF
          + TOK_TABNAME
             + "product_hbase"
          + "p"
       + "="
          + "."
             + TOK_TABLE_OR_COL
                + "a"
             + “prono"
          + "."
             + TOK_TABLE_OR_COL
                + "p"
             + "prono“
```

1

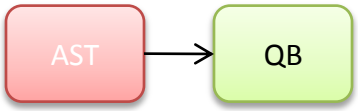
```
+ TOK_INSERT
   + TOK_DESTINATION
      + TOK_TAB
         + TOK_TABNAME
            + "access_log_temp2"
```

2

```
   + TOK_SELECT
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "a"
            + "user"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "a"
            + "prono"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "p"
            + "maker"
      + TOK_SELEXPR
         + "."
            + TOK_TABLE_OR_COL
               + "p"
            + "price"
```

3

Semantic Analyzer (1/3)



AST

```
+ TOK_FROM
   + TOK_JOIN
      + TOK_TABREF
         + TOK_TABNAME
            + "access_log_hbase"
         + a
      + TOK_TABREF
         + TOK_TABNAME
            + "product_hbase"
         + "p"
      + "="
         + "."
            + TOK_TABLE_OR_COL
               + "a"
            + “prono"
         + "."
            + TOK_TABLE_OR_COL
               + "p"
            + "prono“
```

1

QB

MetaData

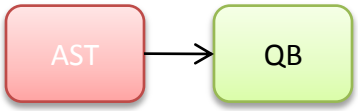
Alias To Table Info
“a”=Table Info(“access_log_hbase”)
“p”=Table Info(“product_hbase”)

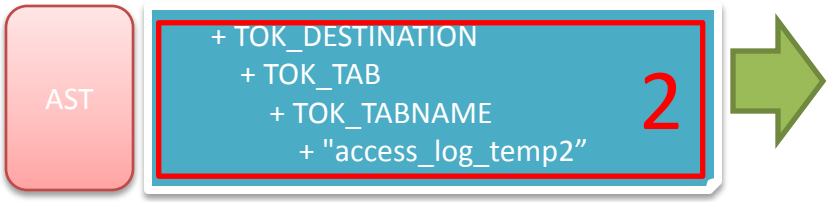
ParseInfo

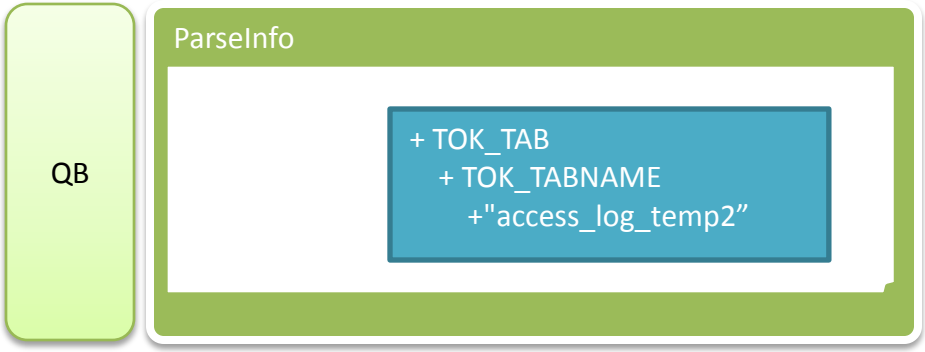
Join Node
+ TOK_JOIN
  + TOK_TABREF
    …
  + TOK_TABREF
    …
  + “=”
    …

Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

Semantic Analyzer (2/3)

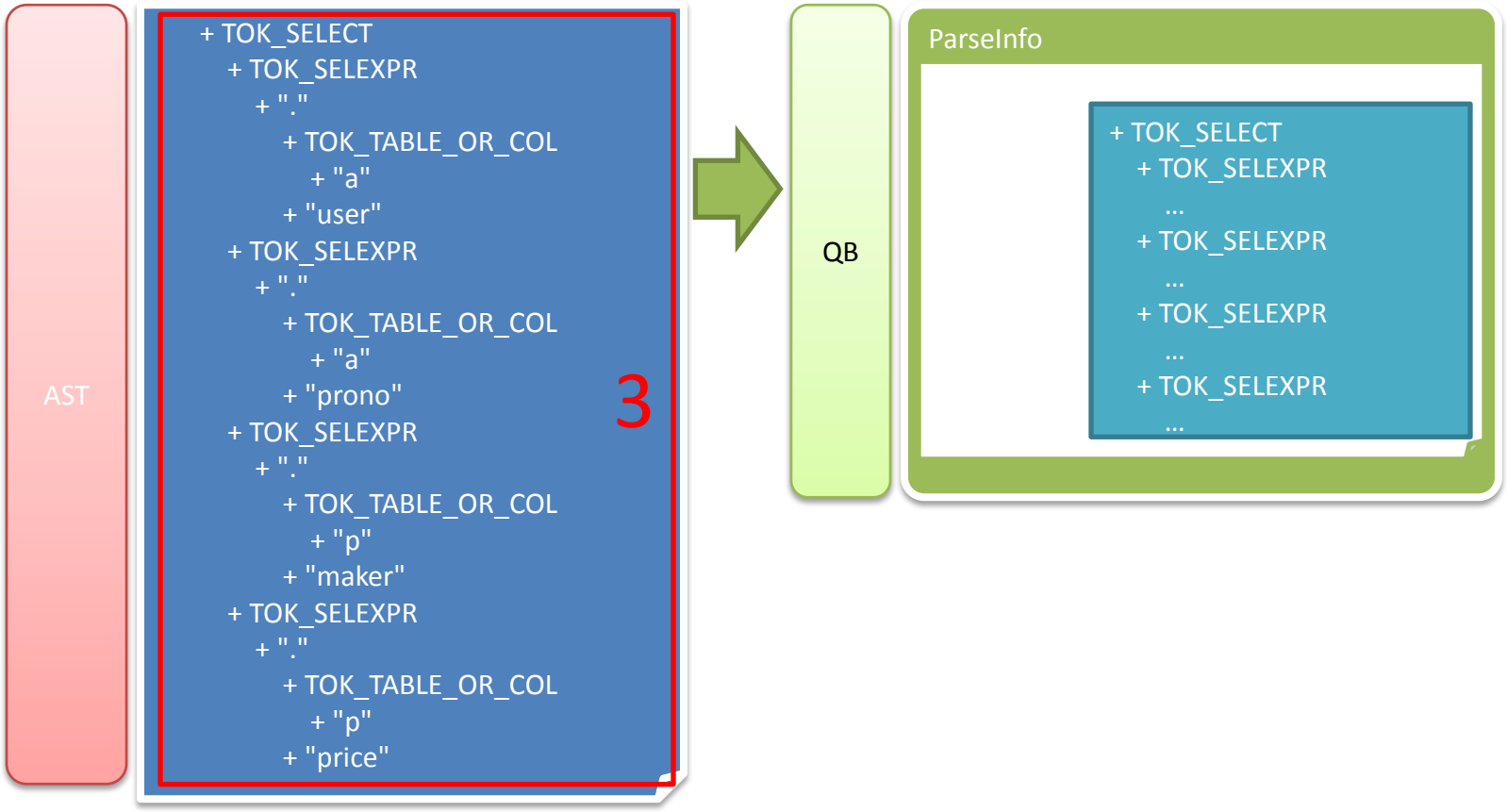






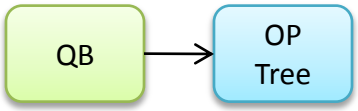
Semantic Analyzer (3/3)

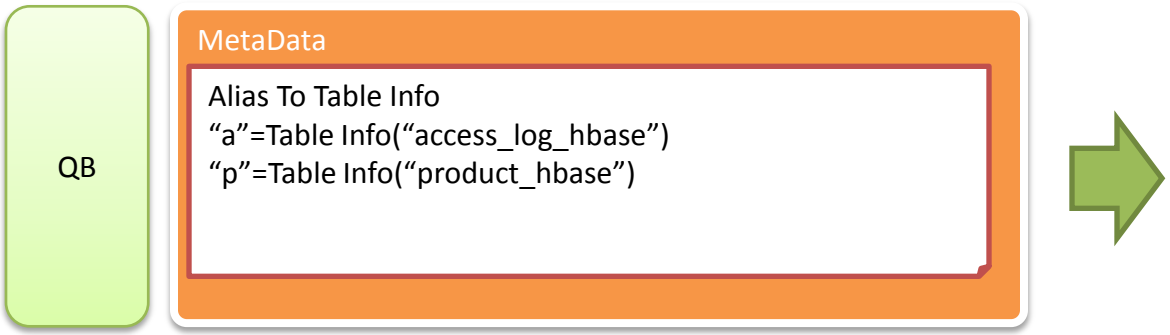
AST → QB

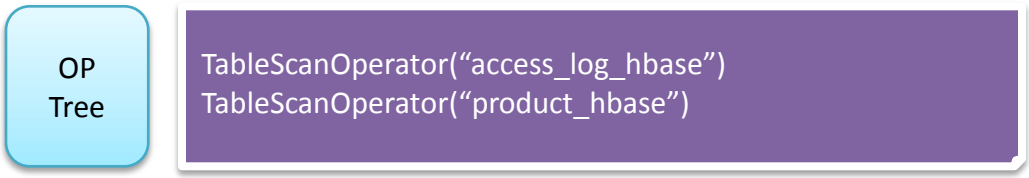


Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

Logical Plan Generator (1/4)

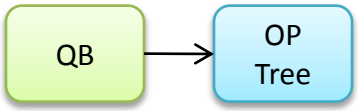






Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

Logical Plan Generator (2/4)



QB

ParseInfo

```
+ TOK_JOIN
    + TOK_TABREF
      + TOK_TABNAME
        + "access_log_hbase"
      + a
    + TOK_TABREF
      + TOK_TABNAME
        + "product_hbase"
      + "p"
    + "="
      + "."
        + TOK_TABLE_OR_COL
          + "a"
        + "access_log_hbase"
      + "."
        + TOK_TABLE_OR_COL
          + "p"
        + "prono“
```

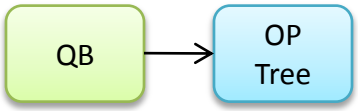
OP Tree

ReduceSinkOperator(“access_log_hbase”)
ReduceSinkOperator(“product_hbase”)

JoinOperator

Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

Logical Plan Generator (3/4)



QB

ParseInfo

Name To Select Node

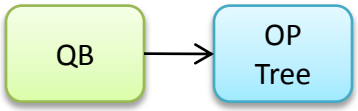
```
+ TOK_SELECT
   + TOK_SELEXPR
      + "."
         + TOK_TABLE_OR_COL
            + "a"
         + "user"
   + TOK_SELEXPR
      + "."
         + TOK_TABLE_OR_COL
            + "a"
         + "prono"
   + TOK_SELEXPR
      + "."
         + TOK_TABLE_OR_COL
            + "p"
         + "maker"
   + TOK_SELEXPR
      + "."
         + TOK_TABLE_OR_COL
            + "p"
         + "price"
```

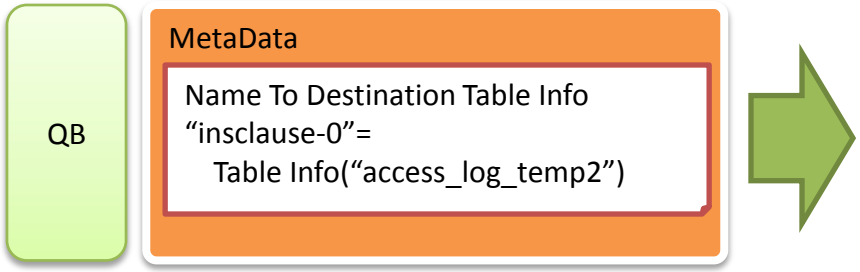
OP Tree

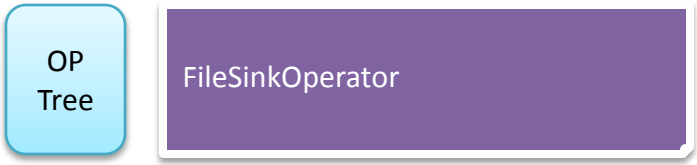
SelectOperator

Parser | Analyzer | Logical Plan Gen. | Logical Optimizer | Physical Plan Gen. | Physical Optimizer

Logical Plan Generator (4/4)



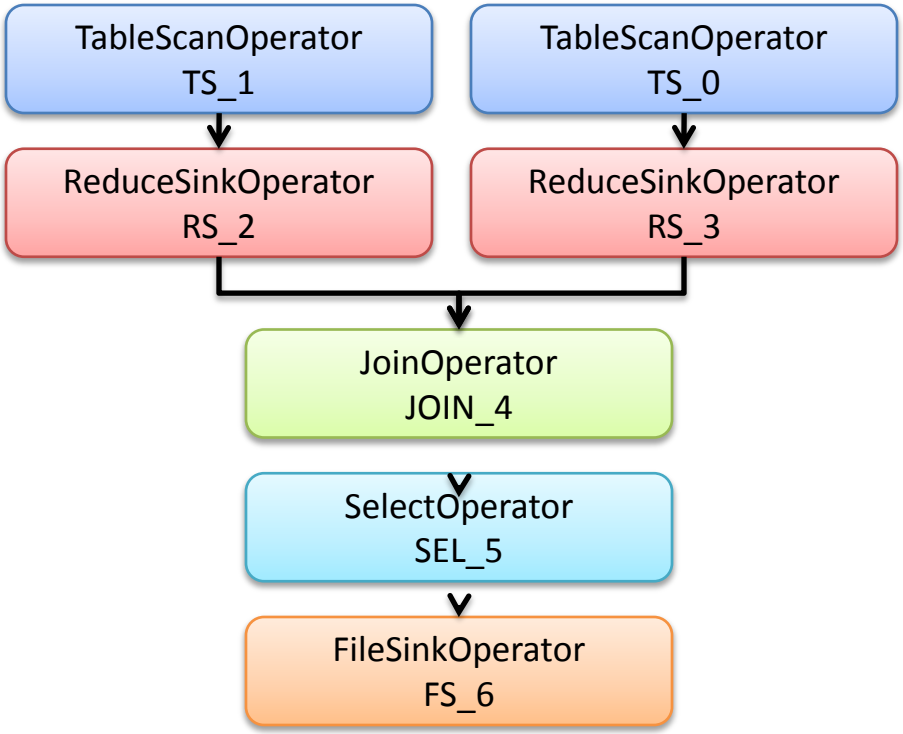




Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

Logical Plan Generator (result)

Op Tree



Logical Optimizer

	说明
LineageGenerator	表与表的血缘关系生成器
ColumnPruner	列裁剪
Predicate PushDown	谓词下推，将只与一张表有关的过滤操作下推至TableScanOperator之后
PartitionPruner	分区裁剪
PartitionCondition Remover	在分区裁剪前，将一些无关的条件谓词去除
GroupByOptimizer	Group By优化
SamplePruner	采样裁剪

	说明
MapJoinProcessor	如果用户指定mapjoin，则将ReduceSinkOperator转换成MapSinkOperator
BucketMapJoin Optimizer	采用分桶的Map Join，扩大Map Join的适用范围
SortedMergeBucket MapJoinOptimizer	Sort Merge Join
UnionProcessor	目前只在两个子查询都是map-only Task时作个标记
JoinReorder	/*+ STREAMTABLE(A) */
ReduceSink DeDuplication	如果两个reduce sink operator共享同一个分区/排序列，则需要对它们进行合并

Logical Optimizer (Predicate Push Down)

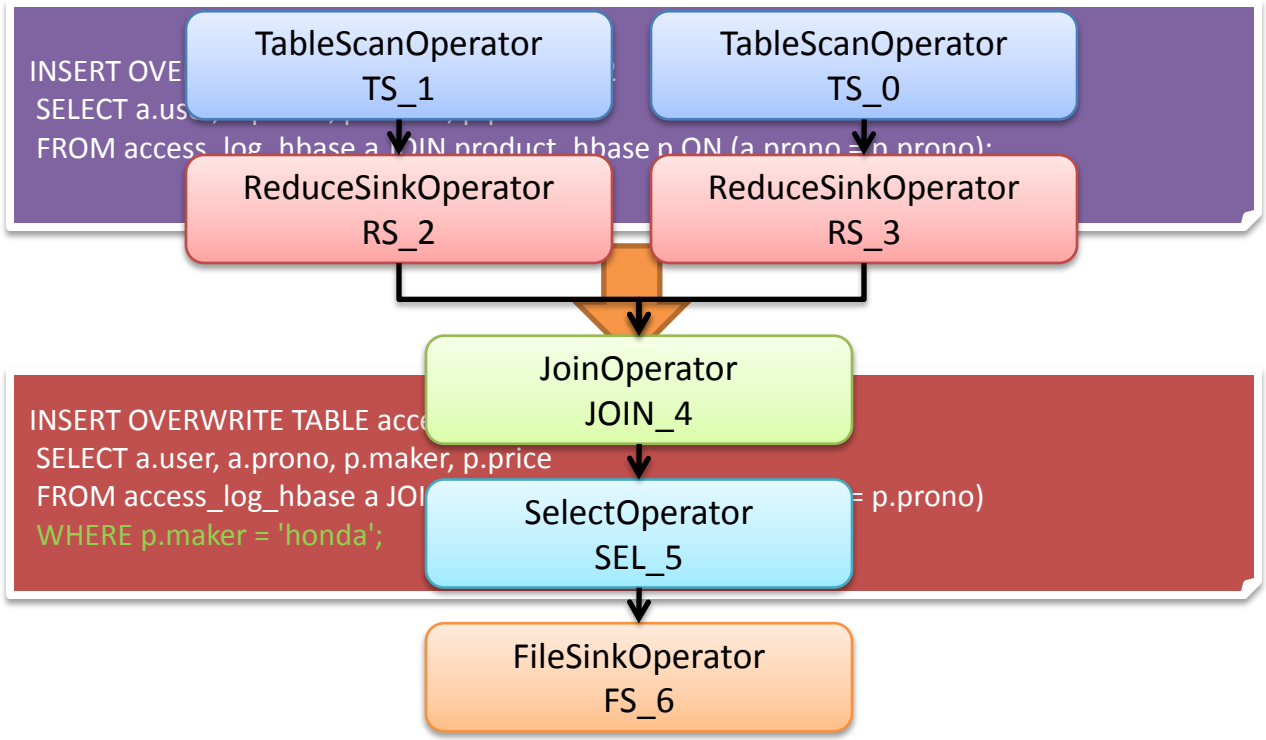
```
INSERT OVERWRITE TABLE access_log_temp2
 SELECT a.user, a.prono, p.maker, p.price
 FROM access_log_hbase a JOIN product_hbase p ON (a.prono = p.prono);
```

Logical Optimizer (Predicate Push Down)

```
INSERT OVERWRITE TABLE access_log_temp2
 SELECT a.user, a.prono, p.maker, p.price
 FROM access_log_hbase a JOIN product_hbase p ON (a.prono = p.prono);
```

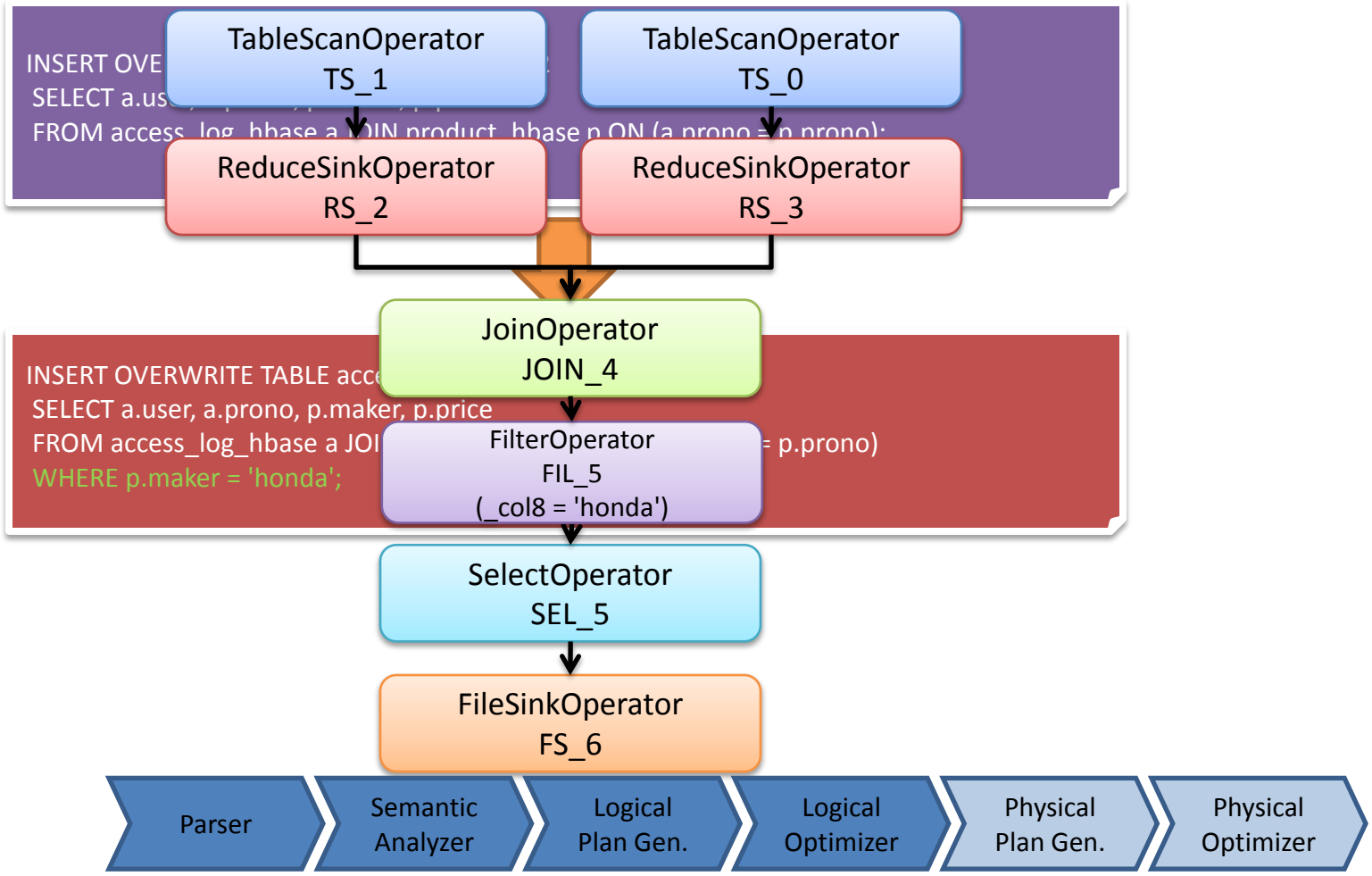
```
INSERT OVERWRITE TABLE access_log_temp2
 SELECT a.user, a.prono, p.maker, p.price
 FROM access_log_hbase a JOIN product_hbase p ON (a.prono = p.prono)
 WHERE p.maker = 'honda';
```

Logical Optimizer (Predicate Push Down)

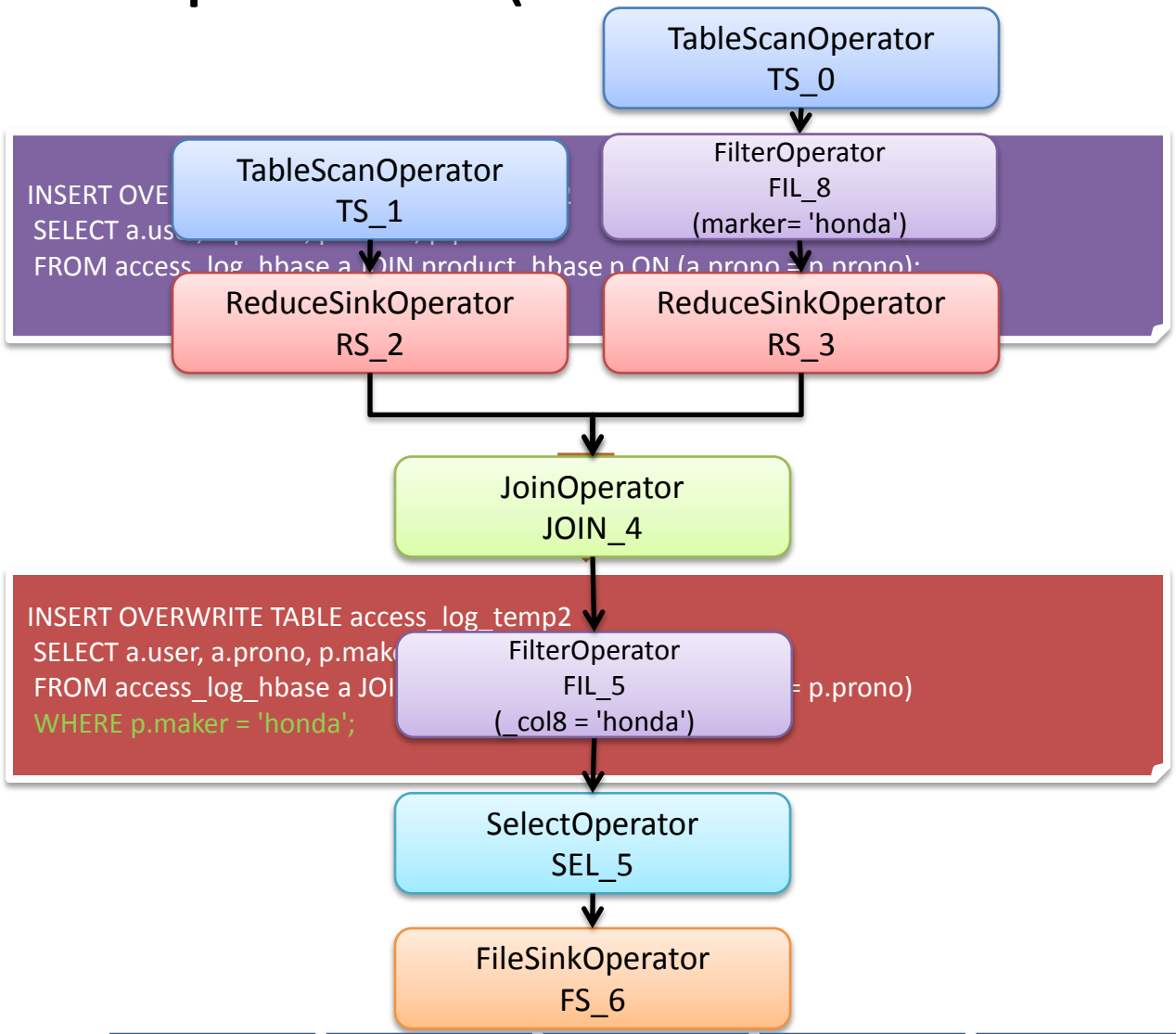


Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

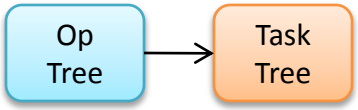
Logical Optimizer (Predicate Push Down)

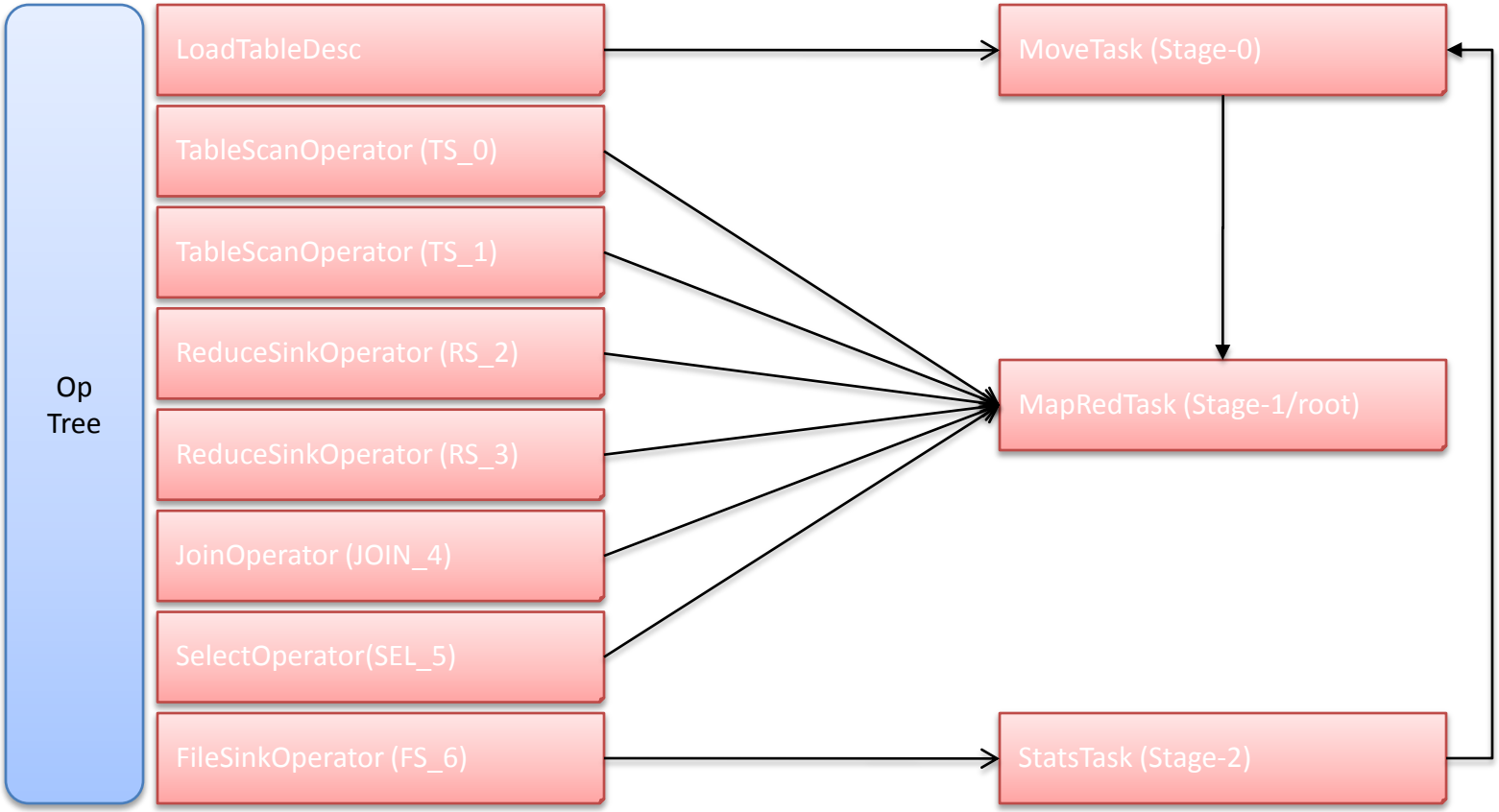


Logical Optimizer (Predicate Push Down)

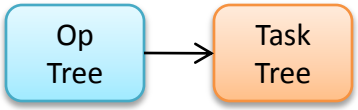


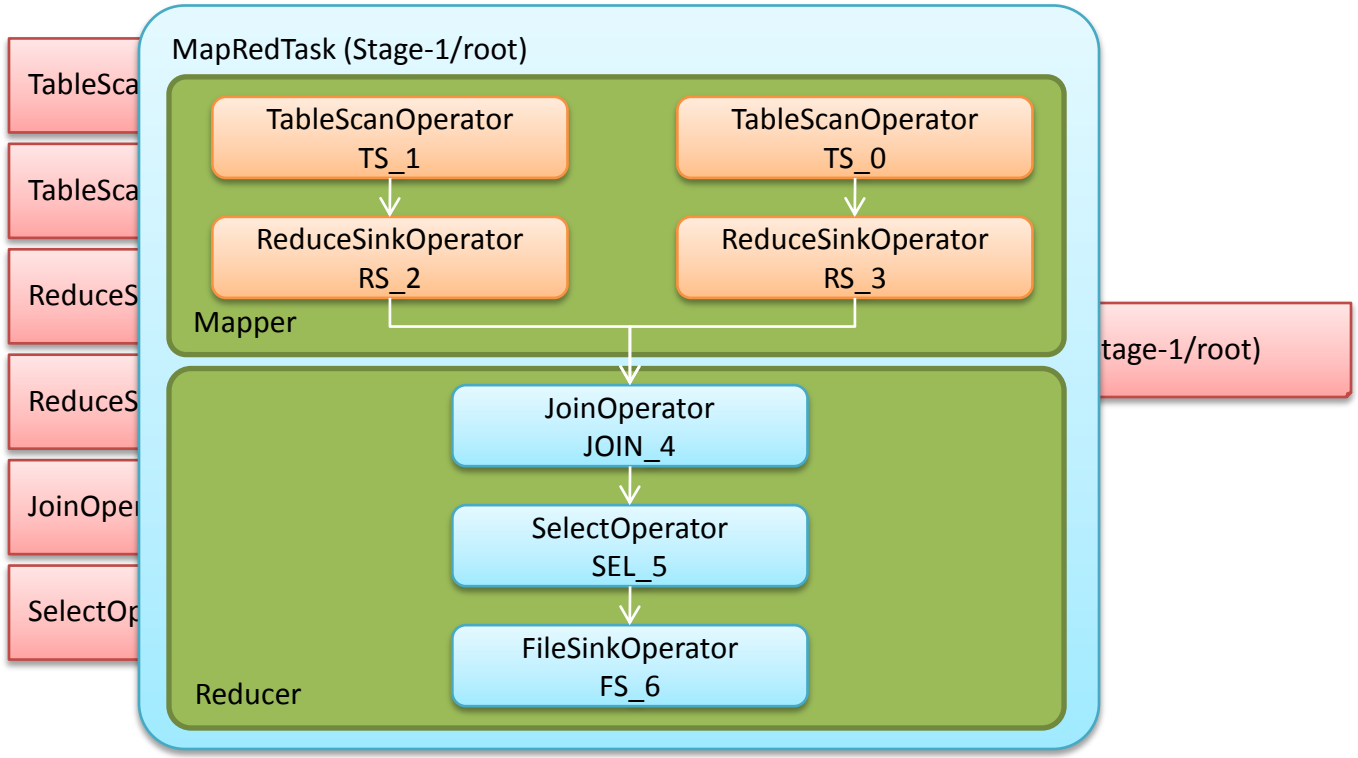
Physical Plan Generator





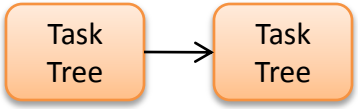
Physical Plan Generator (result)





Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

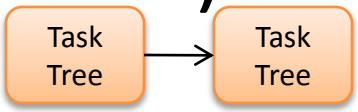
Physical Optimizer

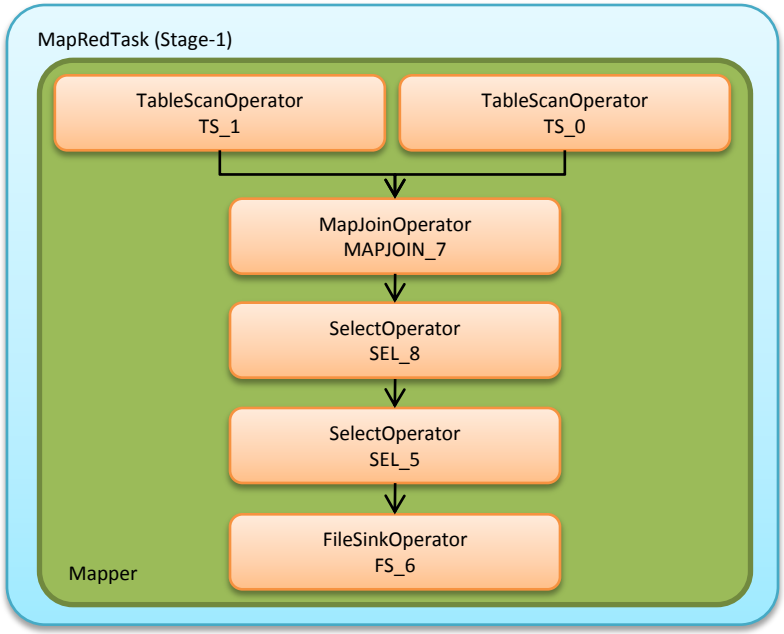


java/org/apache/hadoop/hive/ql/optimizer/physical/目录下

	说明
MapJoinResolver	处理MapJoin
SkewJoinResolver	处理倾斜Join
CommonJoinResolver	处理普通Join

Physical Optimizer (MapJoinResolver)

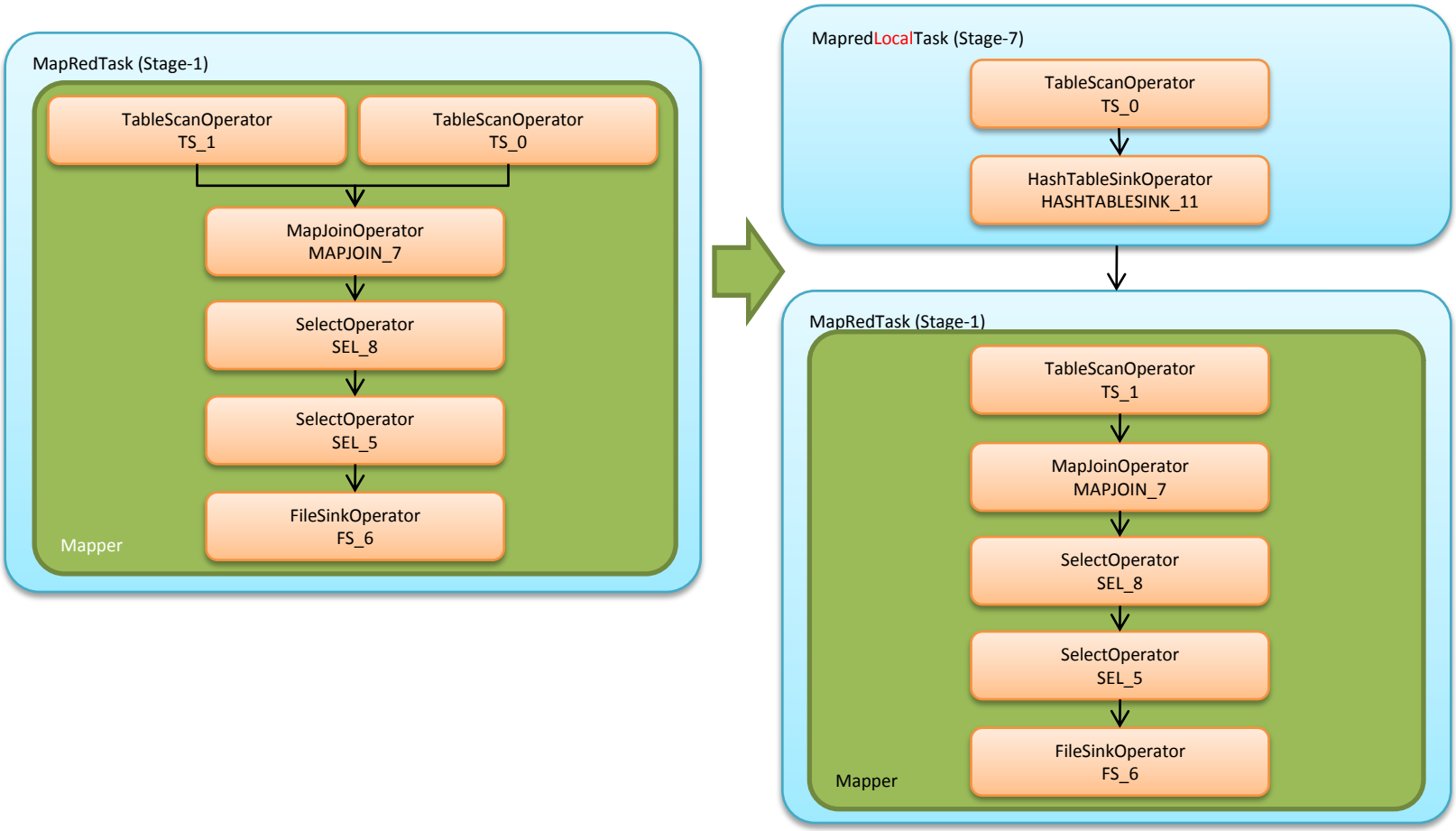




Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

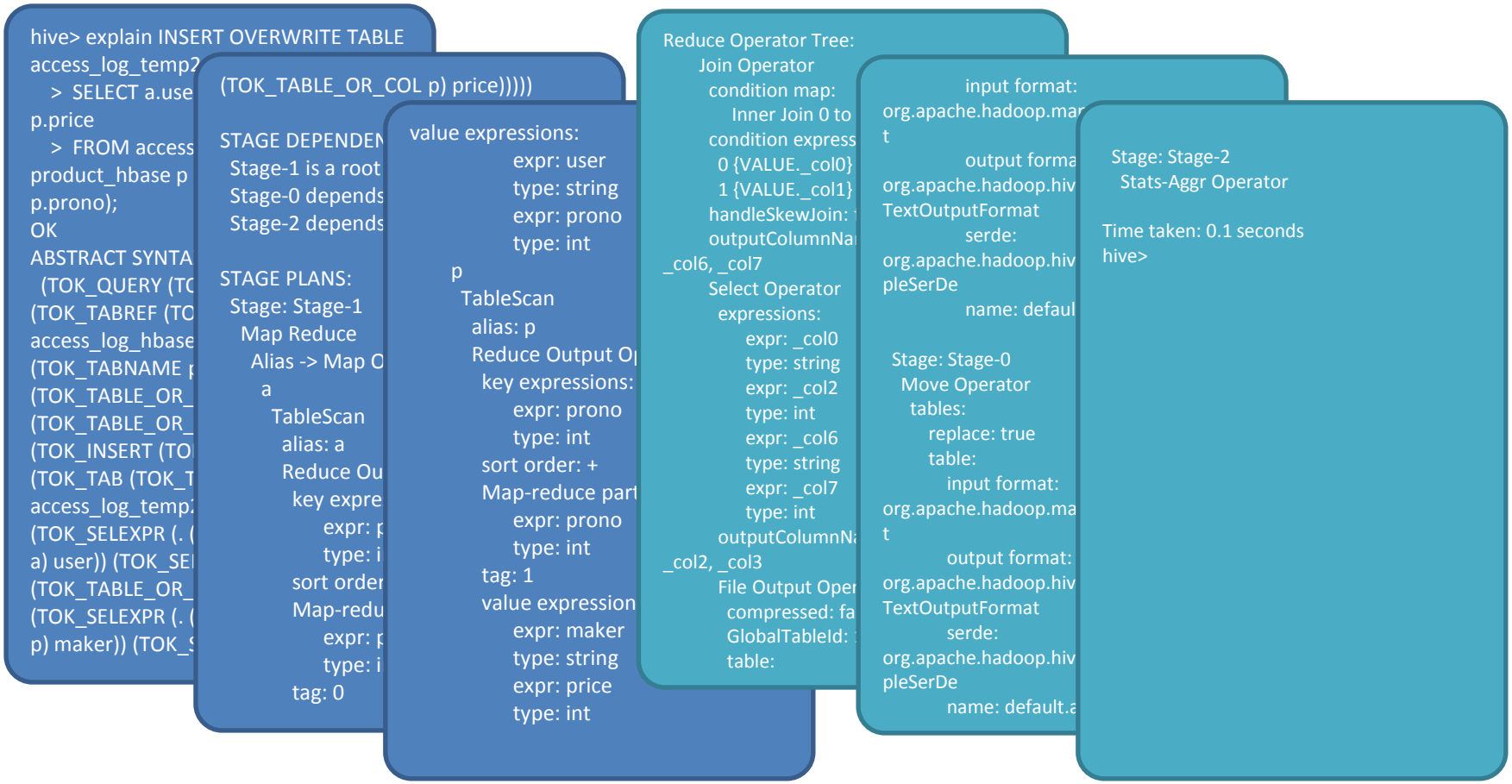
Physical Optimizer (MapJoinResolver)

Task Tree → Task Tree

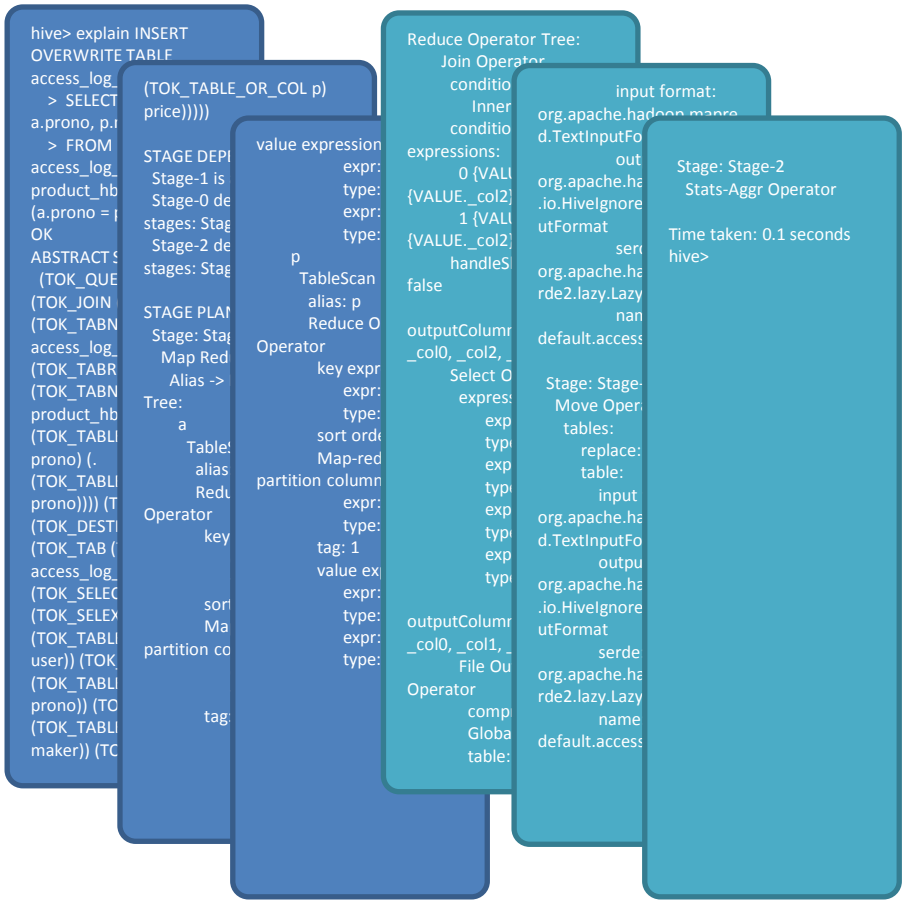


Parser → Semantic Analyzer → Logical Plan Gen. → Logical Optimizer → Physical Plan Gen. → Physical Optimizer

通过Explain观察Hive行为



Hive实现原理



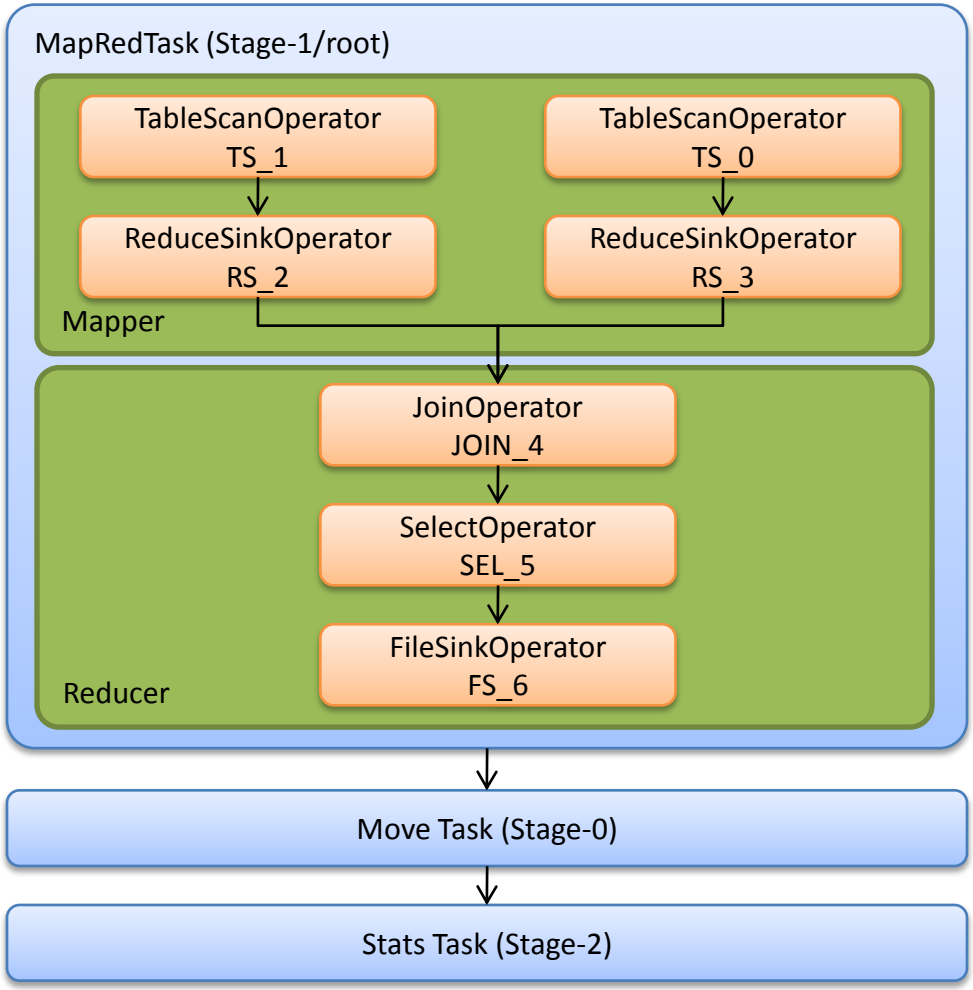
ABSTRACT SYNTAX TREE:

STAGE DEPENDENCIES:
- Stage-1 is a root stage
- Stage-0 depends on stages: Stage-1
- Stage-2 depends on stages: Stage-0

STAGE PLANS:
- Stage: Stage-1
  - Map Reduce
    - Map Operator Tree:
      - TableScan
        - Reduce Output Operator
      - TableScan
        - Reduce Output Operator
    - Reduce Operator Tree:
      - Join Operator
        - Select Operator
          - File Output Operator
- Stage: Stage-0
  - Move Operator
- Stage: Stage-2
  - Stats-Aggr Operator

```
ABSTRACT SYNTAX TREE:

STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1
  Stage-2 depends on stages: Stage-0

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            Reduce Output Operator
          TableScan
            Reduce Output Operator
      Reduce Operator Tree:
        Join Operator
          Select Operator
            File Output Operator

  Stage: Stage-0
    Move Operator

  Stage: Stage-2
    Stats-Aggr Operator
```

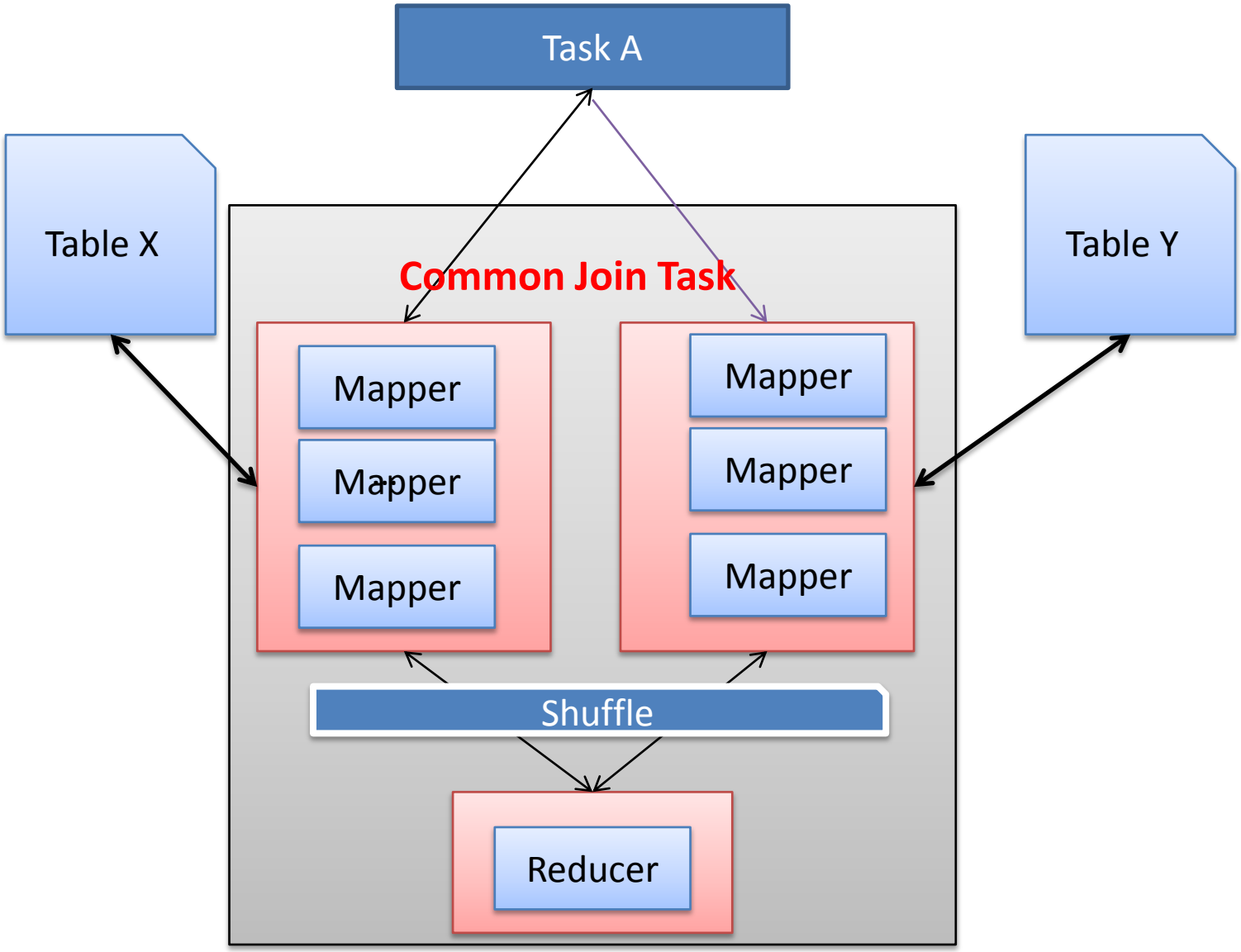
=



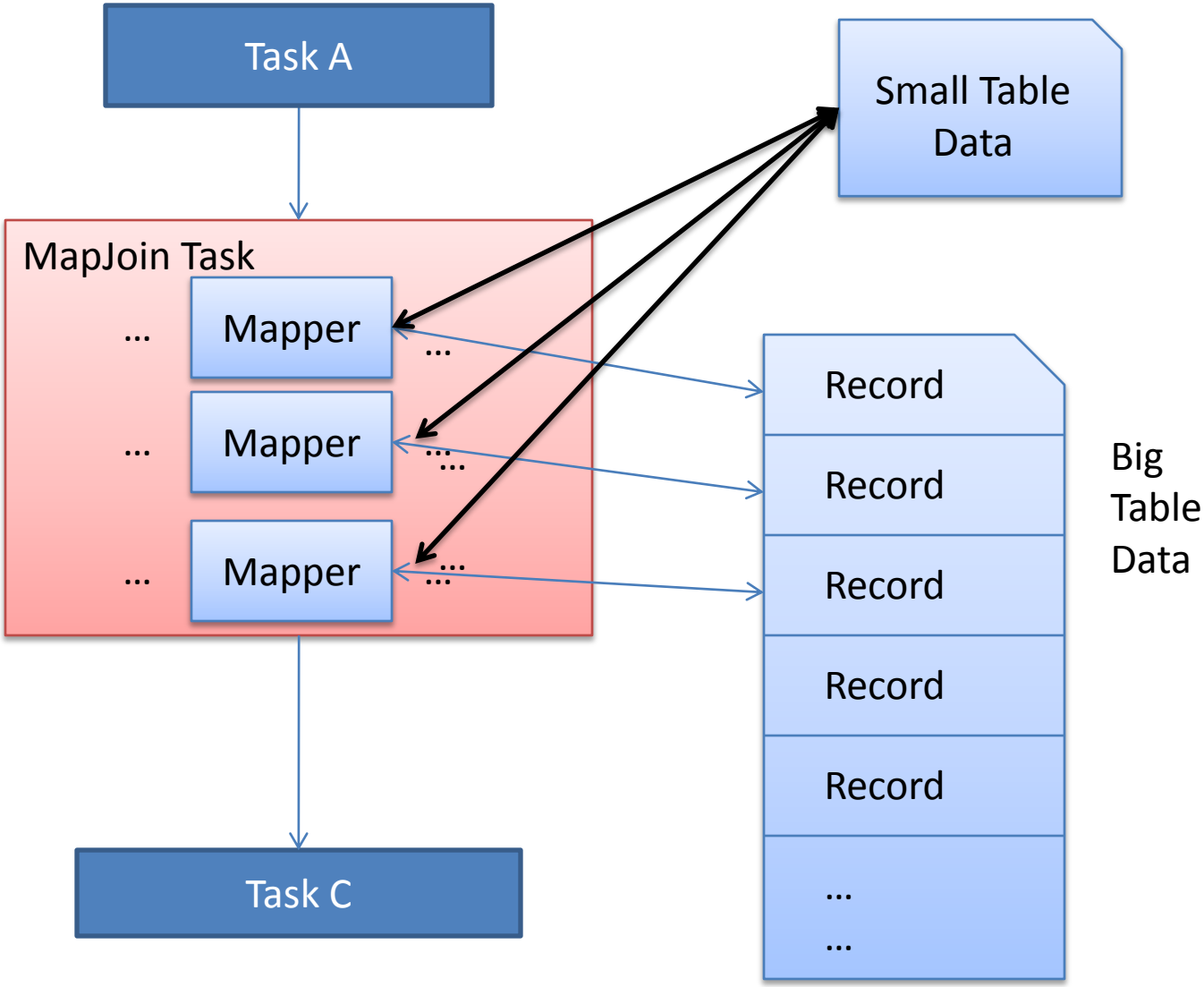
Join的优化


1. E-MAIL PORTFOLIO
CV
2. PHONE INTERVIEW (MULTIPLE)
3. TESTING
4. VISIT Fi (STOCKHOLM OR NY)
5. FACE-TO-FACE INTERVIEW
6. FALL IN LOVE WITH Fi

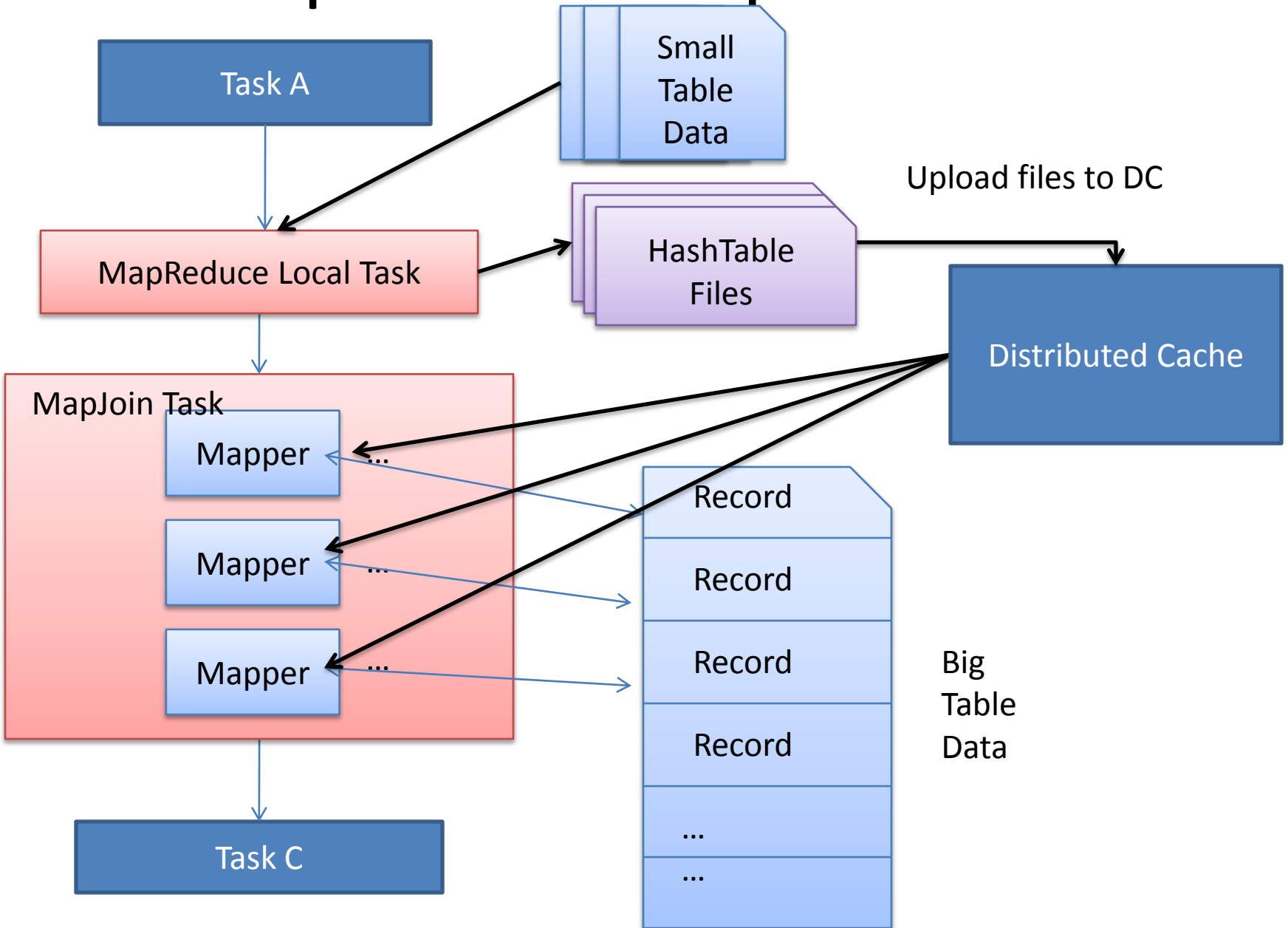
Common Join


Task A
Table X
Table Y
Common Join Task
Mapper
Mapper
Mapper
Mapper
Mapper
Mapper
Shuffle
Reducer

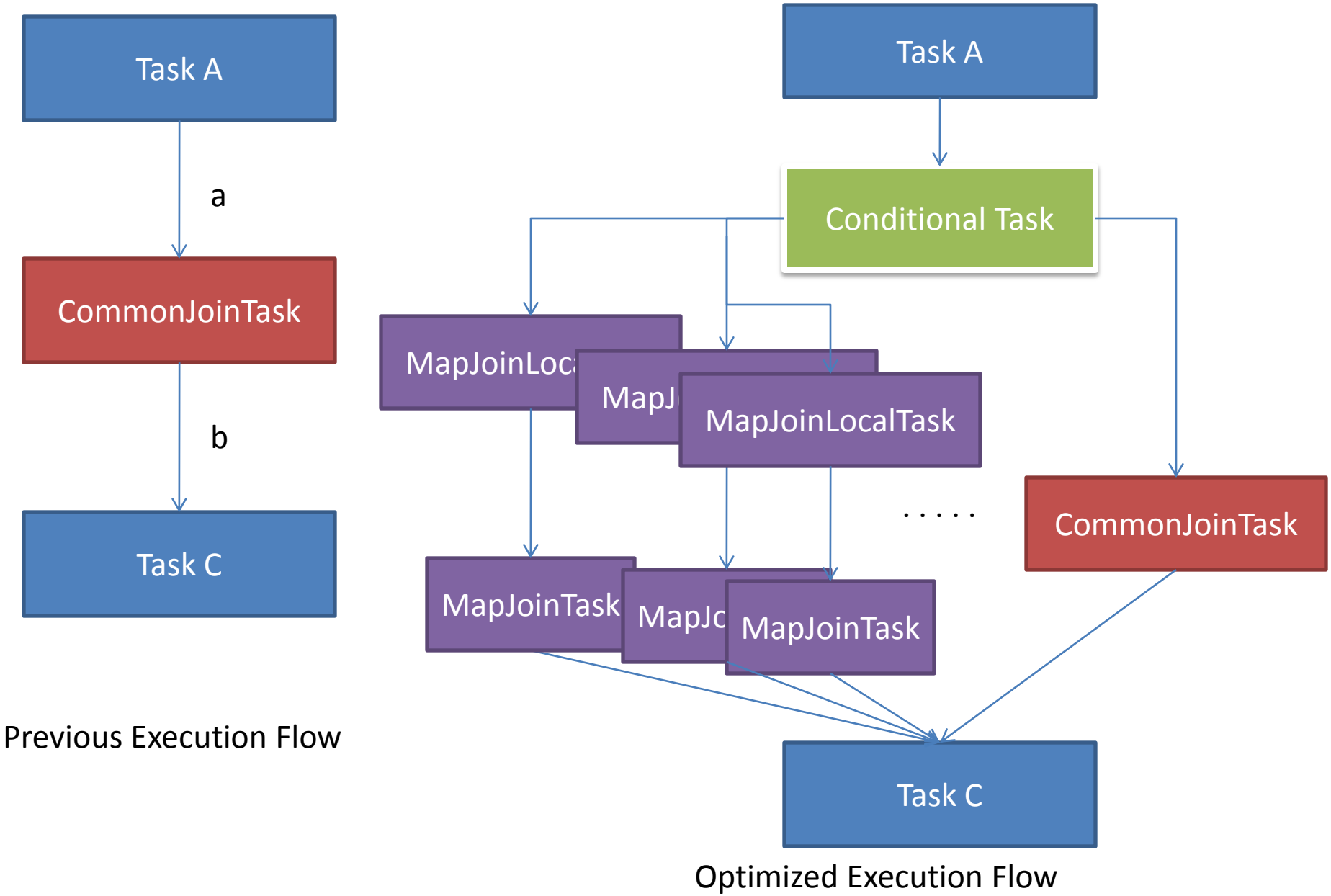
Previous Map Join



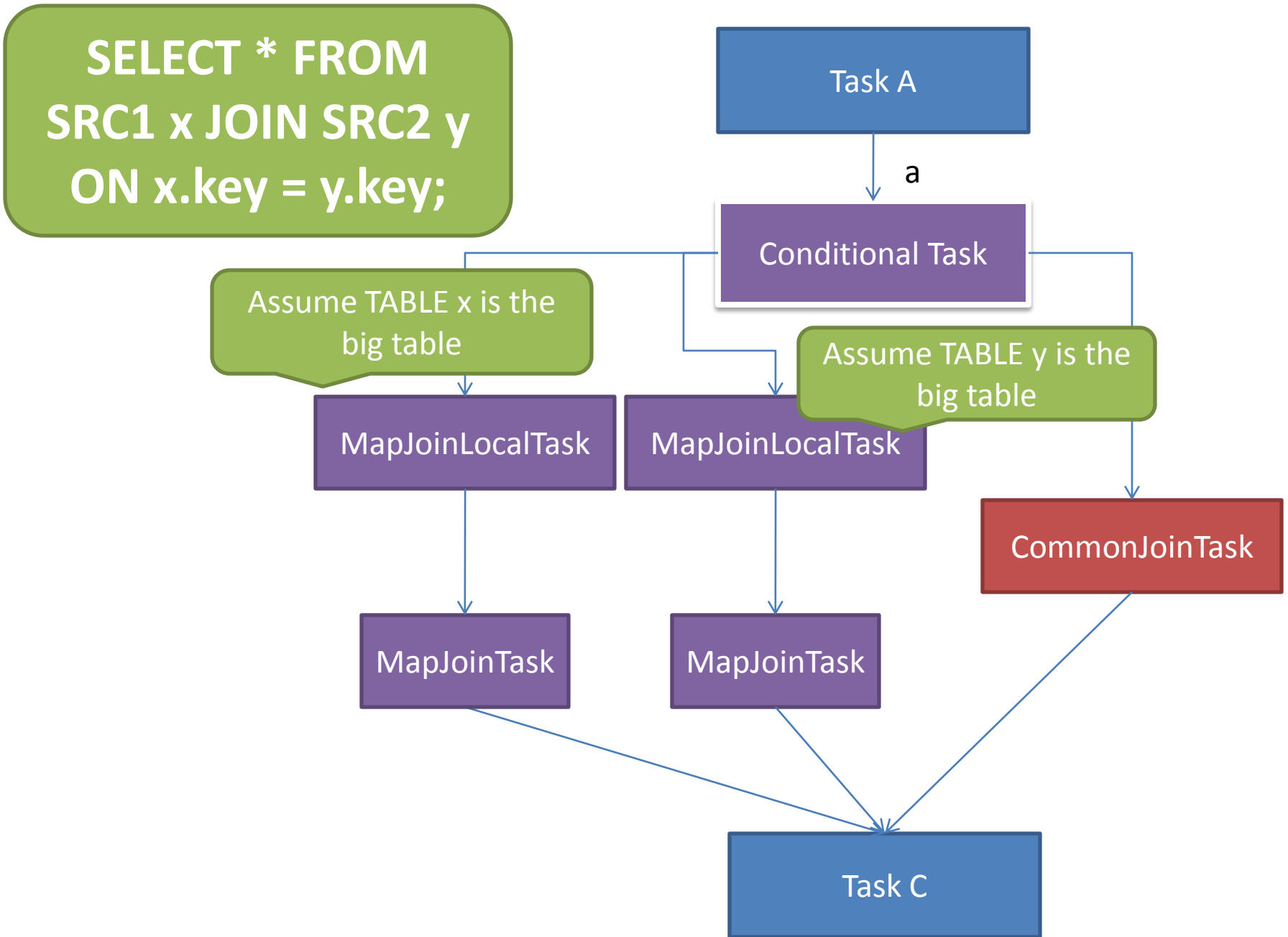
Optimized Map Join



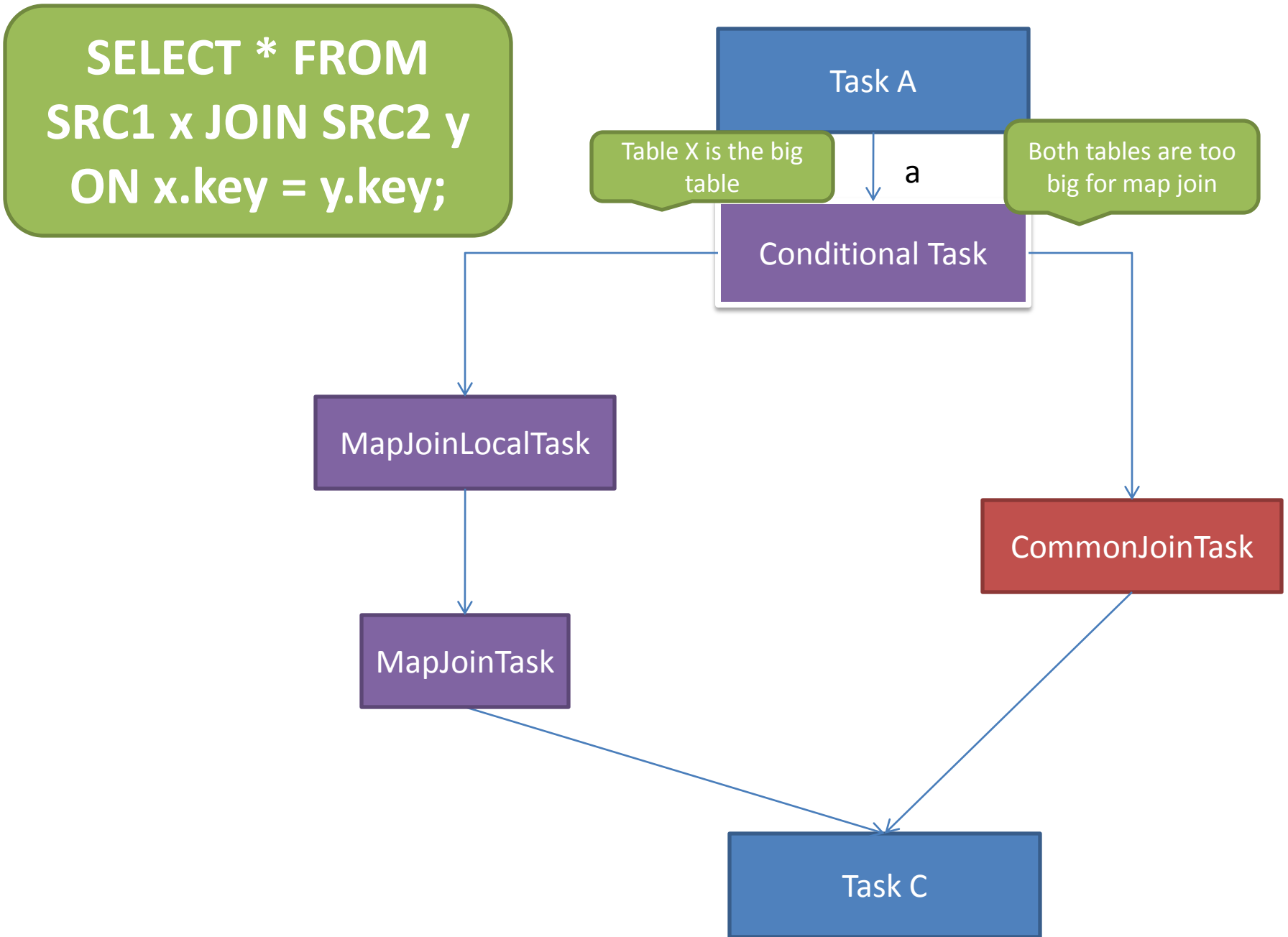
Converting Common Join into Map Join



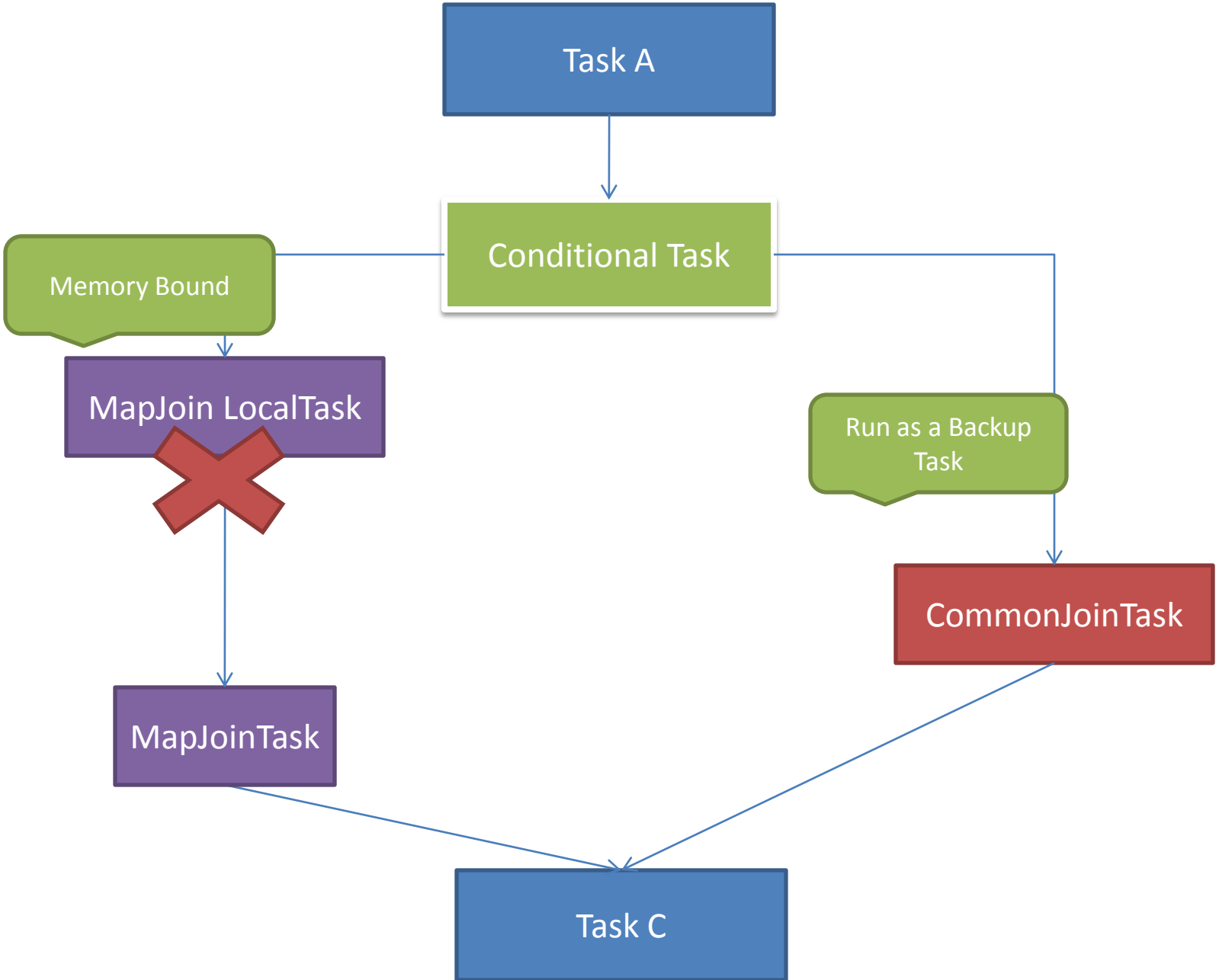
Compile Time



Execution Time



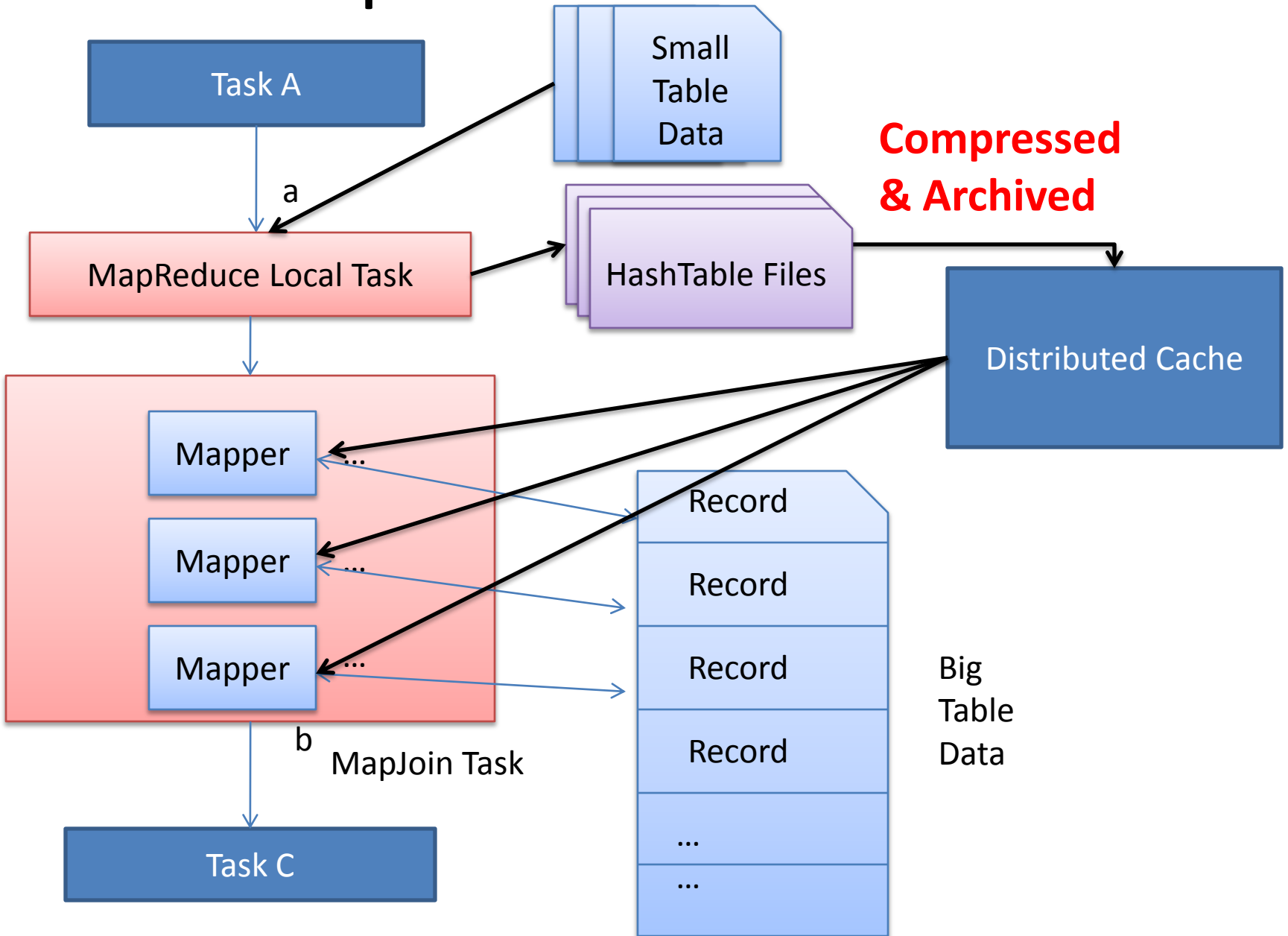
Backup Task


Task A
Conditional Task
Memory Bound
MapJoin LocalTask
Run as a Backup Task
CommonJoinTask
MapJoinTask
Task C

Performance Bottleneck

- Distributed Cache is the potential performance bottleneck
  - Large hashtable file will slow down the propagation of Distributed Cache
  - Mappers are waiting for the hashtables file from Distributed Cache
- Compress and archive all the hashtable file into a tar file.

Compress and Archive



Performance Evaluation

Small Table	Big Table	Join Condition	**Average Join Execution Time Without Compression**	**Average Join Execution Time With Compression**	**Performance Improvement**
75 K rows; 383K file size	130 M rows; 3.5G file size;	1 join key, 2 join value	**106 sec**	**73 sec**	**+ 45%**
500 K rows; 2.6M file size	130 M rows; 3.5G file size	1 join key, 2 join value	**129 sec**	**106 sec**	**+21 %**
75 K rows; 383K file size	16.7 B rows; 459 G file size	1 join key, 2 join value	**441 sec**	**326 sec**	**+ 35 %**
500 K rows; 2.6M file size	16.7 B rows; 459 G file size	1 join key, 2 join value	**326 sec**	**251 sec**	**+30 %**
1M rows; 10M file size	**16.7 B rows; 459 G file size**	**1 join key, 3 join value**	**495 sec**	**266sec**	**+86 %**
1M rows; 10M file size	**16.7 B rows; 459 G file size**	**2 join key, 2 join value**	**425 sec**	**255 sec**	**+67%**

Performance Evaluation

Small Table	Big Table	Join Condition	**Previous Common Join**	**Optimized Common Join**	**Performance Improvement**
75 K rows; 383K file size	130 M rows; 3.5G file size;	1 join key, 2 join value	**169 sec**	**79 sec**	**+ 114%**
500 K rows; 2.6M file size	130 M rows; 3.5G file size	1 join key, 2 join value	**246 sec**	**144 sec**	**+71 %**
75 K rows; 383K file size	16.7 B rows; 459 G file size	1 join key, 2 join value	**511 sec**	**325 sec**	**+ 57 %**
500 K rows; 2.6M file size	16.7 B rows; 459 G file size	1 join key, 2 join value	**502 sec**	**305 sec**	**+64 %**
1M rows; 10M file size	16.7 B rows; 459 G file size	1 join key, 3 join value	**653 sec**	**248 sec**	**+163 %**
1M rows; 10M file size	16.7 B rows; 459 G file size	2 join key, 2 join value	**1117sec**	**536 sec**	**+108%**

Left Semi Join

- 实现 IN/EXISTS 子查询

```
SELECT A.*
FROM A WHERE A.KEY IN
  (SELECT B.KEY FROM B WHERE B.VALUE > 100);
```

等同于:

```
SELECT A.*
FROM A LEFT SEMI JOIN B
  ON (A.KEY = B.KEY and B.VALUE > 100);
```

- 优化
  - map端group by,用来减少流入 reducer端的数据量
  - Join一旦匹配, 立即退出

Bucket Map Join

set hive.optimize.bucketmapjoin = true;

1.和map join一起工作

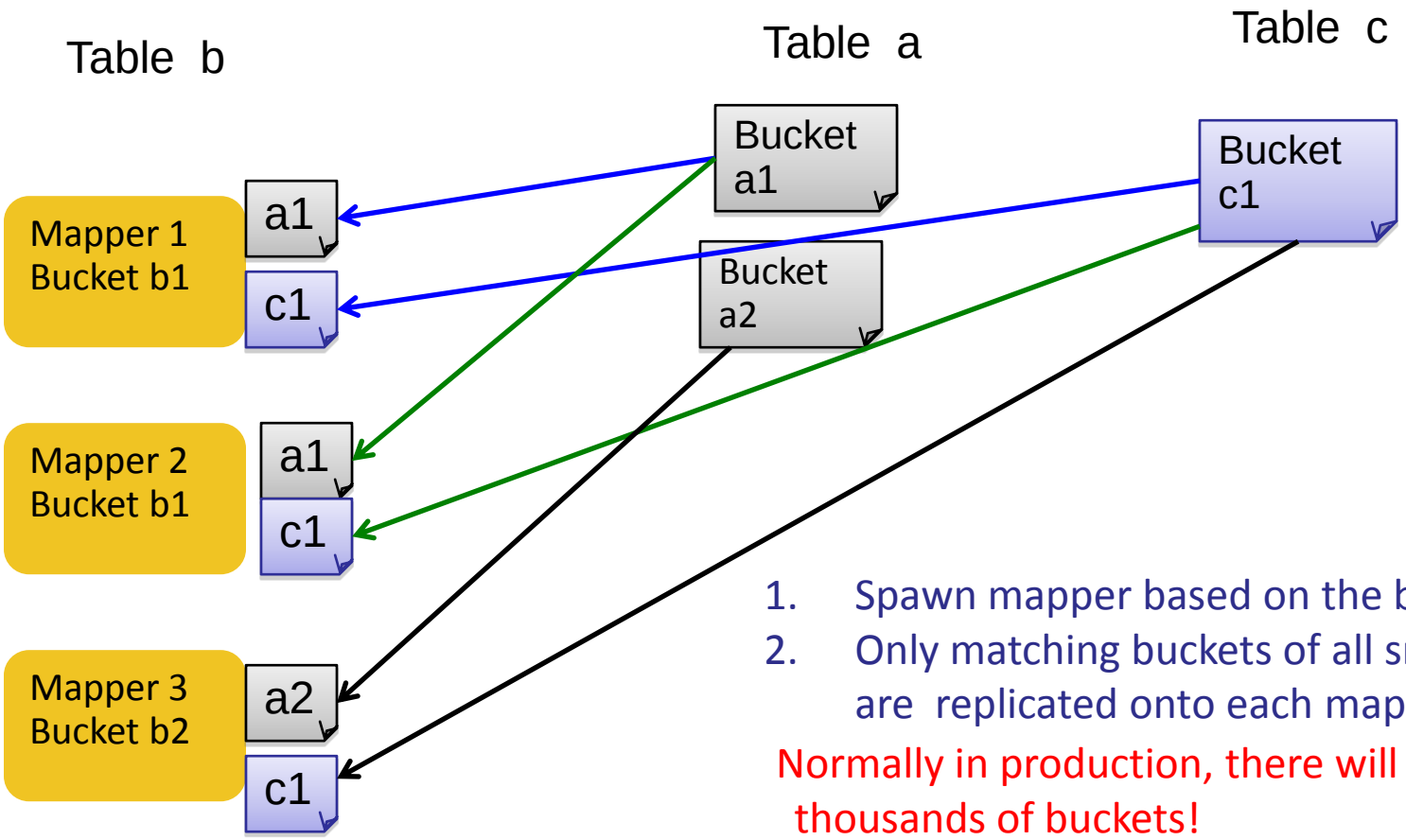
2.所有要join的表都必须做了分桶(bucket)，大表的桶个数是小表桶个数的整数倍.

3.做了bucket的列必须等于join的列

Bucket Map Join实现

```
SELECT /*+MAPJOIN(a,c)*/ a.*, b.*, c.*
a join b on a.key = b.key
 join c on a.key=c.key;
```

Table a,b,c all bucketized by 'key'
a has 2 buckets, b has 2, and c has 1



1. Spawn mapper based on the big table
2. Only matching buckets of all small tables are replicated onto each mapper

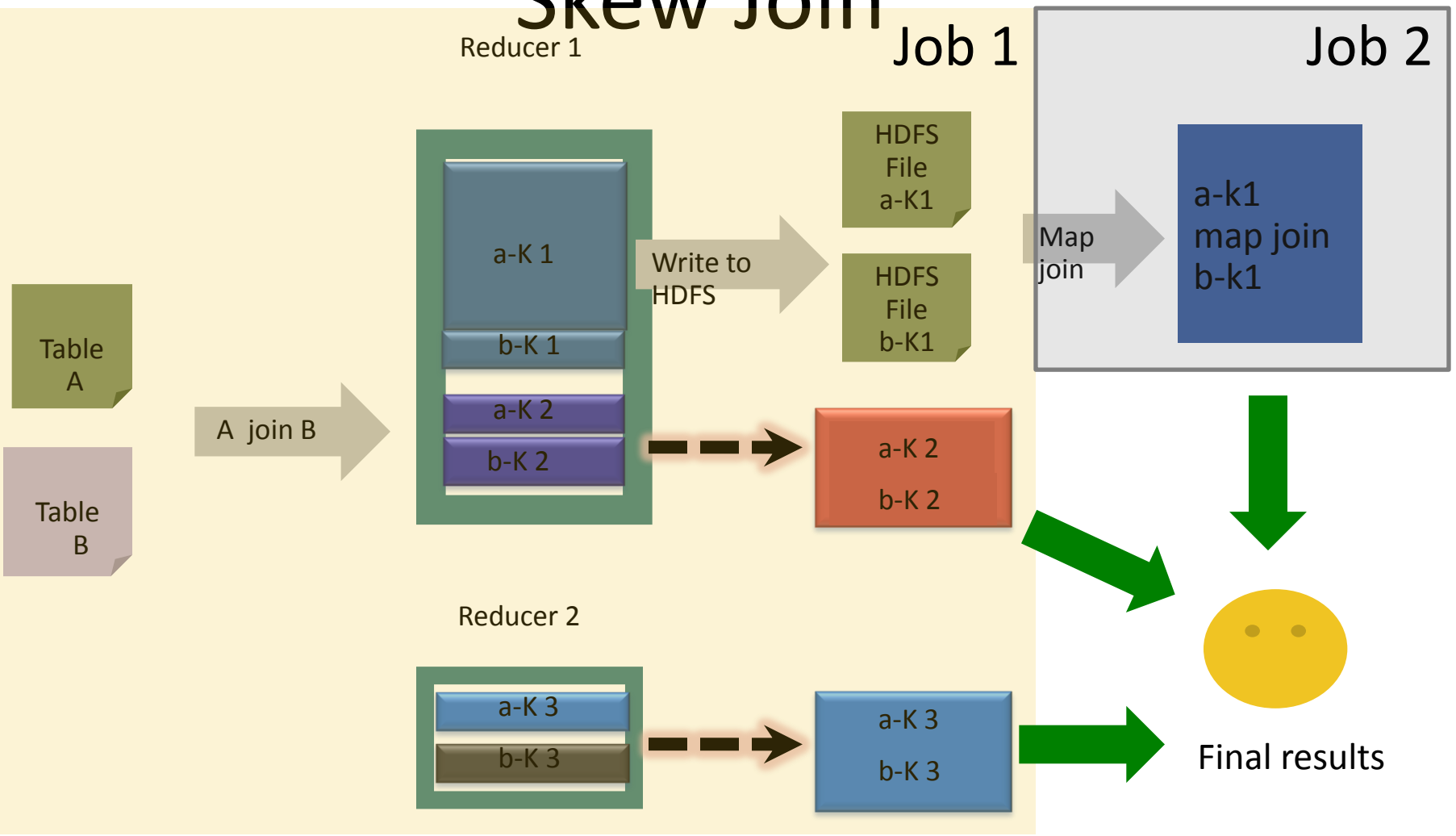
Normally in production, there will be thousands of buckets!

Skew Join

Join时数据倾斜,造成Reduce端OOM

set hive.optimize.skewjoin = true;
set hive.skewjoin.key = 阀值;

Skew Join


Job 1
Job 2
Reducer 1
Table A
Table B
A join B
a-K 1
b-K 1
a-K 2
b-K 2
Write to HDFS
HDFS File a-K1
HDFS File b-K1
Map join
a-k1 map join b-k1
a-K 2
b-K 2
Reducer 2
a-K 3
b-K 3
a-K 3
b-K 3
Final results

常用链接

- Hive官网 http://hive.apache.org
- Wiki https://cwiki.apache.org/confluence/display/Hive/Home
- JIRA https://issues.apache.org/jira/browse/HIVE
- SVN http://svn.apache.org/repos/asf/hive/

Q & A

作者：周忱 | 淘宝综合业务
微博：@MinZhou
邮箱：zhouchen.zm@taobao.com

淘宝网 Taobao.com